

A LIDAR BASED SEMI-AUTONOMOUS COLLISION AVOIDANCE SYSTEM  
AND THE DEVELOPMENT OF A HARDWARE-IN-THE-LOOP SIMULATOR  
TO AID IN ALGORITHM DEVELOPMENT AND HUMAN STUDIES

A Thesis  
presented to  
the Faculty of California Polytechnic State University,  
San Luis Obispo

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

by  
Thomas Fitzgerald Stevens  
December 2015

© 2015

Thomas Fitzgerald Stevens

ALL RIGHTS RESERVED

## COMMITTEE MEMBERSHIP

**TITLE:** A LiDAR Based Semi-Autonomous Collision Avoidance System and the Development of a Hardware-in-the-Loop Simulator to Aid in Algorithm Development and Human Studies

**AUTHOR:** Thomas Fitzgerald Stevens

**DATE SUBMITTED:** December 2015

**COMMITTEE CHAIR:** Dr. Charles B. Birdsong, Ph.D.  
Professor of Mechanical Engineering

**COMMITTEE MEMBER:** Dr. Xiao-Hua (Helen) Yu, Ph.D.  
Professor of Electrical Engineering

**COMMITTEE MEMBER:** Dr. John Fabijanic, Ph.D.  
Lecturer in Mechanical Engineering

## ABSTRACT

# A LiDAR Based Semi-Autonomous Collision Avoidance System and the Development of a Hardware-in-the-Loop Simulator to Aid in Algorithm Development and Human Studies

Thomas Fitzgerald Stevens

In this paper, the architecture and implementation of an embedded controller for a steering based semi-autonomous collision avoidance system on a 1/10<sup>th</sup> scale model is presented. In addition, the development of a 2D hardware-in-the-loop simulator with vehicle dynamics based on the bicycle model is described. The semi-autonomous collision avoidance software is fully contained onboard a single-board computer running embedded GNU/Linux. To eliminate any wired tethers that limit the system's abilities, the driver operates the vehicle at a user-control-station through a wireless Bluetooth interface. The user-control-station is outfitted with a game-controller that provides standard steering wheel and pedal controls along with a television monitor equipped with a wireless video receiver in order to provide a real-time driver's perspective video feed. The hardware-in-the-loop simulator was developed in order to aid in the evaluation and further development of the semi-autonomous collision avoidance algorithms. In addition, a post analysis tool was created to numerically and visually inspect the controller's responses. The ultimate goal of this project was to create a wireless 1/10<sup>th</sup> scale collision avoidance research platform to facilitate human studies surrounding driver assistance and active safety systems in automobiles. This thesis is a continuation of work done by numerous Cal Poly undergraduate and graduate students.

## ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Birdsong, for his continued support and guidance throughout my studies and thesis work. His expertise, insight, and supportive remarks always opened new avenues in which to venture. I extend my thanks for the opportunity to work on this exciting and relevant project that has led to me pursue a career in ADAS.

I would like to thank my committee members, Dr. Yu and Dr. Fabijanic, for their help and support. I would also like to thank my parents for their unceasing encouragement and support. I am grateful for the previous teams who laid the framework for this thesis work through development of the initial semi-autonomous collision avoidance research platform.

## TABLE OF CONTENTS

	Page
LIST OF TABLES .....	ix
LIST OF FIGURES .....	x
NOMENCLATURE .....	xii
Chapter 1 Introduction & Background.....	1
1.1    Introduction and Motivation for Vehicle Research.....	1
1.2    Safety, Cost, and Scaled Vehicle Testing .....	4
1.3    Simulation as a Development Tool .....	6
1.4    Related Work and Current Technologies .....	13
1.5    Previous Work and Goals of This Study .....	17
1.6    Overview of Remaining Chapters .....	20
Chapter 2 Theory.....	22
2.1    Fundamentals of Vehicle Dynamics .....	22
2.1.1    Bicycle Model .....	25
2.1.2    Tire Models and Lateral Forces.....	30
2.1.3    Engine and Drivetrain Modeling and Power Limited Acceleration.....	40
2.1.4    Static Loads and Longitudinal Load Transfer on Level Ground.....	42
2.1.5    Rolling Resistance .....	43
2.1.6    Aerodynamic Drag .....	44
2.1.7    Low-Speed versus High-Speed Model.....	44
2.1.8    Numerical Integration.....	45
2.1.9    Steering Methods for Path-Tracking .....	46
2.1.10    Segmentation and Threat Determination.....	48
2.1.11    RRT and Path-planning .....	50
2.2    Fundamentals of Real-Time Embedded Systems.....	53
2.2.1    Finite-State Machines, Multithreading, and Thread Safety .....	55
2.3    Fundamentals of Simulation and GUI Development .....	59
2.3.1    Java Swing and Java2D API .....	59
2.3.2    Active vs. Passive Rendering .....	60
2.3.3    Image Blitting.....	61

2.4	Pin Multiplexing and Device Tree Overlays.....	61
2.5	PWM and Servo Control.....	63
2.6	UART Serial Communication.....	63
Chapter 3	Developing the Semi-Autonomous Collision Avoidance System and an Embedded Vehicle Controller.....	65
3.1	Single-Board Computer Selection and Evaluation.....	67
3.2	Porting of existing code from Matlab/Simulink to C++ .....	71
3.3	Embedded System Software Architecture.....	73
3.4	Embedded System Hardware .....	86
3.5	LiDAR .....	90
3.6	Pulse-Width Modulation (PWM) Motor Control.....	92
3.7	Bluetooth Serial Communication .....	95
3.8	Wheel Encoders.....	97
3.9	User Control Development and Simulation Interfacing.....	98
3.10	BeagleBone Specific Issues and Recompiling the Linux Kernel.....	101
3.11	Running the System and Modes of Operation .....	103
3.11.1	Simulation Mode .....	103
3.11.2	Free Roam Bluetooth Driver Mode (Open-Loop).....	104
3.11.3	Full System Semi-Autonomous Collision Avoidance Mode (Closed-Loop).....	104
3.11.4	Writing Linux Scripts and Services.....	104
Chapter 4	Developing a Hardware-In-Loop Simulation .....	106
4.1	Simulator Design.....	108
4.2	Motor Dynamic Simulation and Parameters .....	115
4.3	Vehicle Model Interface with Embedded Controller .....	117
4.4	Creating and Rendering a Simulated Environment.....	117
4.5	GUI, User Control Development, and Simulation Interfacing.....	119
4.6	PID Speed Controller and Primitive Vehicle AI .....	120
Chapter 5	Testing and Results.....	122
5.1	Tire Tests and Tire Testing Machine .....	122
5.2	Vehicle Dynamometer.....	127

5.3	Simulator Post Analysis Tool.....	129
5.4	Hardware-in-the-Loop Simulator Testing.....	135
5.4.1	Previous ESV Algorithm.....	135
5.4.2	LiDAR's Resolution Influence on Controller Performance.....	136
5.4.3	Reducing Unjustified Controller Overrides By Improving Threat Determination .....	138
5.4.4	Following the Same Path between Iterations of the RRT Algorithm.....	140
5.5	Embedded Controller Testing .....	144
Chapter 6	Conclusion and Recommendations .....	150
BIBLIOGRAPHY .....		158
APPENDICES		
Appendix A	Hardware-in-the-Loop Simulation Controls .....	162
Appendix B	Embedded System Code.....	163
Appendix C	Hardware-in-the-Loop Simulation (VDHILS) Code .....	261
Appendix D	User Control Station (UCS) Code.....	508
Appendix E	Post Analysis Tool .....	596
Appendix F	Development Board Specification Matrix.....	668

## LIST OF TABLES

Table	Page
Table 1. Bill of Materials (BOM) .....	90
Table 2. Power classes of Bluetooth modules .....	96
Table 3. Parameters of Traxxas RC truck.....	123

## LIST OF FIGURES

Figure	Page
Figure 1. Ohio University Driving Simulator .....	8
Figure 2. Automated Dynamics Analysis of Mechanical Systems (ADAMS/Car) Screenshot .....	9
Figure 3. The Open Racing Car Simulator (TORCS) Screenshot .....	10
Figure 4. CarSim Software Screenshot .....	11
Figure 5. Prescan Simulation Software.....	12
Figure 6. Electronic Stability Control (ESC)/Electronic Stability Program (ESP) Active Safety System.....	15
Figure 7. Mercedes Braking System.....	16
Figure 8. Google Self-Driving Car .....	17
Figure 9. The 1/10th Scale Semi-Autonomous Vehicle Platform and Control Loop .....	19
Figure 10. SAE Vehicle Axis System.....	24
Figure 11. The Dynamic Bicycle Model.....	27
Figure 12. Forces Acting on the (Steerable) Front and Rear Tire .....	30
Figure 13. Lateral force versus slip angle .....	31
Figure 14. Mechanism of tire lateral force in elastic range .....	33
Figure 15. Example Magic Formula curve .....	36
Figure 16. Circle of forces, traction circle, friction circle, or friction ellipse concept .....	37
Figure 17. Calspan Tire Research Facility, TIRF .....	39
Figure 18. Cornering Stiffness Testing Apparatus .....	39
Figure 19. Performance characteristics of gasoline and diesel engines .....	41
Figure 20. Pure-Pursuit Geometry .....	48
Figure 21. Lidar scan (blue) with parsed obstacles (black rectangles) and photo of scanned area.....	50
Figure 22. RRT nodes and branches mapping valid paths (green) and failed paths (red) in the driving plane.....	50
Figure 23. Sample of the results of the RRT algorithm.....	52
Figure 24. Example State Transition Diagram for a Turnstile Finite-State Machine .....	56
Figure 25. Shared memory model.....	59
Figure 26. Example of bit blitting used in computer graphics.....	61
Figure 27. Example Pin Muxing or Multiplexing Diagram.....	62
Figure 28. BeagleBone Black SBC and features .....	69
Figure 29. Matlab/Simulink semi-autonomous collision avoidance software model.....	72
Figure 30. Real-time embedded software task diagram.....	76
Figure 31. Monitor Bluetooth FSM state transition diagram.....	80
Figure 32. Motor updater FSM state transition diagram.....	81
Figure 33. Poll encoders FSM state transition diagram.....	82

Figure 34. Poll encoder FSM state transition diagram .....	84
Figure 35. Avoid collisions FSM state transition diagram .....	85
Figure 36. System hardware architecture.....	87
Figure 37. BeagleBone cape board design and populated board .....	88
Figure 38. 1/10th Semi-Autonomous Collision Avoidance Platform.....	89
Figure 39. Hokuyo UBG-04LX-F01 LiDAR sensor .....	91
Figure 40. Traxxas XL-5 Motor ESC .....	92
Figure 41. Driver and Controller PWM Scaling.....	93
Figure 42. Throttle to PWM Scaling.....	94
Figure 43. Steer Angle to PWM Scaling .....	94
Figure 44. Sparkfun Bluetooth Mate Gold .....	97
Figure 45. QRE1113 Line Sensor .....	98
Figure 46. User-control-station and GUI .....	99
Figure 47. Logitech joystick with steering wheel and foot pedals .....	100
Figure 48. SkyZone 5.8GHz 200mW FPV Wireless AV Tx & Rx Set.....	101
Figure 49. Ladder of Abstraction.....	110
Figure 50. Delineation of Reality and Simulation .....	111
Figure 51. Simple game-loop/simulation-loop .....	114
Figure 52. Multithreaded game-loop/simulation-loop with time regulation .....	115
Figure 53. DC Motor Model Power Curves.....	116
Figure 54. Road tiles and example generated road maps.....	118
Figure 55. Example Path Generation for an SLC Maneuver .....	120
Figure 56. Skid-pad (upper right) and tire dynamometer tests (upper left, lower left, lower right) .....	125
Figure 57. Tire dynamometer and skid-pad test results performed with varying vertical loads ( $F_z$ ) .....	126
Figure 58. Vehicle dynamometer used in conjunction with the simulation .....	129
Figure 59. Example of post analysis tool's output .....	132
Figure 60. Brake assistance functionality when no safe path can be found and an imminent threat exists (cyan curves show failed paths, no valid paths present, brake percentage shown in left panel) .....	134
Figure 61. Previously implemented tethered system .....	137
Figure 62. Wireless system implementation.....	138
Figure 63. Threat determination diagram with respect to road boundaries .....	140
Figure 64. From left-to-right, low threat region example, medium threat region example, and high threat region example .....	140
Figure 65. Subsequently planned paths around an obstacle showing cause of oscillatory steer commands.....	142
Figure 66. Controller's open and closed-loop CPU load and memory use .....	147
Figure 67. Path-planning with artificial potential fields .....	153

## NOMENCLATURE

$x, y$	Longitudinal and lateral position, respectively, in the vehicle frame
$V_x, V_y$	Longitudinal and lateral velocities, respectively, in the vehicle frame
$\dot{V}_x, \dot{V}_y$	Longitudinal and lateral accelerations, respectively, in the vehicle frame
$\psi$	Yaw angle in the vehicle frame
$\dot{\psi}$	Yaw rate in the vehicle frame
$\ddot{\psi}$	Yaw angular acceleration in the vehicle frame
$X, Y$	Longitudinal and lateral position in the global frame
$F_x, F_y$	Longitudinal and lateral forces, respectively, in vehicle frame
$F_{xf}, F_{yf}$	Longitudinal and lateral forces, respectively, acting on front tire
$F_{xr}, F_{yr}$	Longitudinal and lateral forces, respectively, acting on rear tire
$R_{xf}, R_{xr}$	Rolloing resistance acting on the front and rear tire, respectively
$F_{\text{drag}}$	Aerodynamic drag force
$M_z$	Moment about the z-axis in the vehicle frame
$\omega$	Angular velocity of the tire
$m$	Total vehicle mass
$I_z$	Total vehicle moment of inertia about the z-axis
$g$	Acceleration due to gravity
$L$	Vehicle wheel base
$a$	Distance from vehicle's center of gravity to the front tire
$b$	Distance from vehicle's center of gravity to the rear tire
$h$	Distance from vehicle's center of gravity to the ground
$R$	Tire radius
$G$	Power-train gearing ratio
$T_m$	Motor torque
$\omega_m$	Angular velocity of the motor
$R_a$	Armature resistance
$K_b$	Back EMF constant
$K_t$	Motor torque constant
$E_a$	Applied armature voltage
$\alpha_f, \alpha_r$	Local side slip angle at the front and rear tire, respectively
$\alpha_f, \alpha_r$	Global side slip angle at the front and rear tire, respectively
$\delta$	Steer angle
$C_\alpha$	Tire cornering stiffness
$l_d$	Pure-pursuit control law look forward distance
$\mu$	Friction coefficient
$W$	Total vehicle weight
$W_f, W_r$	Weight on the front and rear tire, respectively

$C_D$	Coefficient of drag
A	Projected area
$f_r$	Rolling resistance coefficient

## **Chapter 1     Introduction & Background**

### 1.1        Introduction and Motivation for Vehicle Research

The study of vehicle safety began almost immediately following the invention of the modern automobile. The modern automobile is one of the most complex and dangerous machines that mankind has developed. In 1769, Nicholas Joseph Cugnot made history by fabricating and driving the first road vehicle. Subsequently, he also made history by being involved in one of the first road vehicle accidents (Brennan, 1997) (Gillespie). In 2010, the National Highway Traffic Safety Administration (NHTSA) reported that there were 1,542,000 automotive collisions in the United States alone that resulted in 30,196 fatalities and varying degrees of injury. Nearly 70% of all fatal crashes in 2010 involved frontal collisions and over 10% of these fatal crashes involved distracted drivers (National Highway Traffic Safety Administration, 2010). In 2012, NHTSA estimates that over 34,000 people died in motor vehicle crashes with over 10% involving distracted drivers where the obstacle was never seen by the driver. This is a 5.3% increase compared with 2011 (Michielsen, 2013). Although the number of injuries and fatalities per traveled mile has leveled off in recent decades and is at an all-time low in the USA (National Highway Traffic Safety Administration, 2010), collision avoidance technology is the low-hanging fruit that could significantly lower these statistics further and, if implemented widely, could have an overarching impact on society. Within the automotive industry, Palopoli states that a majority 80% of innovation is accounted for by electronic components and control systems (Palopoli). As Brennan suggests, improvements in crashworthiness of vehicles is at a point of diminishing return while the

field of automated collision avoidance is in its infancy; injuries and fatalities resulting from vehicle collisions could become obsolete altogether.

NHTSA has classified the degree of vehicle autonomy into 5 levels. In level 0, the driver always remains in complete control over the vehicle. Level 1 possesses individually automated controls, such as electronic stability control (ESC). Within level 2, at least two controls are automated in unison, such as adaptive cruise control in combination with lane keep assistance. In level 3, the driver can fully cede safety critical control to the vehicle. The vehicle deems when is appropriate for the driver to retake control and provides sufficiently comfortable transition time. Level 4 vehicles always perform all safety critical functions and the driver is never expected to take control. An important aspect of autonomous active safety systems is their need to electrically control the steering, throttle, and/or brakes in order to override driver control in a high-risk emergency situation. The recent advent of steer-by-wire driver control removes the direct mechanical link between the driver and the vehicle. In this type of system, the driver's commands are interpreted by a computer in order to control and actuate the steering and throttle motors. Steer-by-wire control along with developments in onboard sensing, lane detection, and obstacle recognition facilitate the development and implementation of autonomous or semi-autonomous navigation and collision avoidance systems that would be much more cumbersome with the traditional purely mechanical control mechanisms (Anderson, Peters, Pilutti, Tseng, & Iagnemma) (Weilkes, Burkle, Rentschler, & Scherl, 2005). The first production vehicle to feature a steer-by-wire system is the Infiniti Q50 (Davies, 2014). This system offers a quicker and more precise steering response than its mechanical counterpart, reduces vibrations, and allows for easier integration of

autonomous functions. Vehicles such as this have enabled a tremendous spur in autonomous driving, path-planning, and collision avoidance research along with system development to create intelligent or smart vehicles. For driver assistance or semi-autonomous technologies the driver is kept in the loop and these technologies must account for driver behavior, which is very complex and difficult to model. However, according to Brennan, driver assistance and semi-autonomous technologies provide a natural prelude to complete automation. The research presented in this paper takes this suggestion by developing a semi-autonomous driver collision avoidance system that will serve as a research platform and test-bed for algorithm development and human factor studies. This provides a step towards fully autonomous vehicles while allowing the driver to remain in control of the vehicle until intervention is required and operates on a minimal intervention principle.

Cars are a part of America's culture and are how many people commute. Many see their vehicle as an extension of themselves. In a Pew survey conducted in 2006, it was shown that 69% of people like to drive. Many said it provided them with relaxation, scenery, and freedom (Taylor, Funk, & Craighill). Although drivers are typically decent controllers they are far from impeccable and in emergency situations a computer can process much more environmental data and cues in a much shorter time period to make much more informed decisions. Developing semi-autonomous collision avoidance systems as a first step towards fully autonomous driver assistance technologies is likely to ease adoption of this technology by auto manufacturers and by insurance companies as an attentive driver is still behind the wheel. A semi-autonomous approach could also help to ease people's apprehension to autonomous technologies by reducing injuries &

fatalities and would allow fully autonomous technologies to be more widely accepted upon implementation (Brennan, 1997). To bridge the gap between theory and a marketable product, vehicle automation is likely to gradually develop through driver assistance technologies and change from a more passive to a more active role. Despite the fact that the introduction of this technology may enable people to become distracted or become over-reliant on system intervention, the crash demographics for this system include all frontal and angular collisions with other motor vehicles, collisions with obstacles along the roadway such as telephone poles, and collisions with static obstacles in the path of the vehicle. This demographic represents over 3.8 million collisions per year in the United States alone (National Highway Traffic Safety Administration, 2009) (Stevens, Birdsong, Painter, & Carlson, 2013).

## 1.2 Safety, Cost, and Scaled Vehicle Testing

Developing a controller that can improve performance and replace the driver in all emergency situations can be difficult to verify. The term performance as it pertains to this research refers to improving the safety of the vehicle by using a controller to manipulate the control signals to avoid imminent collisions. As noted by Brennan, it is difficult to create a controller that is as adaptable and intelligent as a human and until a controller can verifiably outperform a human under all circumstances there will likely be strong resistance to vehicle automation. In addition, drivers may be apprehensive of the thought of losing control over their vehicle to a computer even if only for a moment. This problem lends itself to scaled vehicle testing as a safe and cost-effective alternative to develop the controller's algorithms and to investigate driver's perception of relinquished control without risking their safety.

Scaled vehicle testing possesses many advantages over full-scale testing including cost, development time, and safety to drivers, observers, and hardware. The sensory equipment used on scaled vehicles may be much more limited in terms of size, frequency, and range which further reduces cost and decreases development time. In addition, this allows many widely available off-the-self radio controlled (RC) vehicles that already feature a steer-by-wire control configuration to be purchased instead of building custom components from the ground up. This allows a controller to be developed and readily incorporated in between the driver and the actuated control commands. Many of the sensors used in full scale driver assistance and autonomous technologies are not suitable for scaled vehicles. However, the field of robotics has prompted the need for small high-precision sensors. Many relatively low-cost, high-resolution sensors with small footprints have come to market, namely light detection and ranging (LiDAR) sensors. For scaled vehicles with much fewer vehicle components and subsystems, the time required to change or modify the vehicle and/or software is greatly reduced when compared to a full size vehicle. Without the infrastructure to investigate fleet wide autonomous vehicle implementation and intelligent vehicle highway based systems, steps must be taken to develop a controller that can reliably perform better than a human driver in emergency circumstances (Brennan, 1997). The research of semi-autonomous and fully-autonomous vehicle controllers fit well within the realm of university sponsored research efforts to develop the corresponding algorithms and investigate human factors surrounding this technology. This will likely be very useful to auto manufacturers in the coming years as this technology is slowly adopted into the US fleet.

### 1.3 Simulation as a Development Tool

Even by utilizing a scaled vehicle for testing and development, a great deal of time is still required to set-up and test the system for each change to the vehicle or software. These efforts may be irrespective of the larger picture of algorithm development and human studies and could dilute observations during development. In order to develop the software and hardware used in autonomous vehicles, computer simulation is a vital tool that can be used to develop and collision mitigation algorithms in a safe environment without the capital costs of designing, developing, and setting up a test rig. The testing and verification is of crucial importance to ensure that functionality is correct; simulation is an effective means to reduce time-consuming debugging in the field and allows for much more iteration in the design process. Simulation allows for running thousands of different test scenarios in a fraction of the time it takes to run full-scale system tests. Not only does simulation allow reduced capital costs, but when using a graphical simulation one may visualize the results of these tests and provides both qualitative and quantitative judgments of the degree of performance of the controller. In addition, the user can better tune algorithm parameters and observe their influence on the overall system performance with small investments in time and cost further driving improved quality. Time can be focused on generating the best algorithmic solutions even before experimental field testing.

Many high-fidelity 3D dynamic car simulators exist that have dynamic models that are very realistic in terms of vehicle dynamics and the driving experience. These simulators and models represent huge efforts in order to capture the extremely complex non-linear effects that influence a vehicle's dynamic behaviors. Typically, these models

contain thousands of parameters that may be individually altered to model a particular vehicle or driving scenario. These simulators may be interfaced with custom vehicle controllers in order to observe their effect on vehicle performance with striking realism. These simulators are not limited to driver technology development but are also used extensively in realistic video-game and racing simulators where drivers expect a realistic driving experience. Many of these simulators are proprietary and target specific audiences for particular uses. Some examples include the Ohio State University Driving Simulator, ADAMS/Car, TORCS, CarSim, and PreScan. A brief description of each simulator follows.

Ohio University houses one of only 40 such driving simulators around the world which is used to determine the most effective traffic devices for highway operations and work zone safety and can be used to help assess the cause of roadway crashes (Ohio University, 2008). This simulator allows researchers to overcome limited statistically significant sample sizes to test traffic situations on hundreds of volunteer drivers. The lab features a half-cab Ford Focus that is connected to a computer simulation capable of creating hundreds of different driving scenarios, from rural roads dotted with deer and emergency vehicles to icy highways choked with traffic. This simulator is also capable of tracking a driver's eye-movements and monitoring facial features to measure how long a driver look at specific objects and can ascertain the alertness or distraction of the driver.



Figure 1. Ohio University Driving Simulator

Automated Dynamics Analysis of Mechanical Systems (ADAMS) software is a multibody dynamics simulator that allows engineering teams to quickly build and test functional virtual prototypes of vehicles and vehicle subsystems increasing productivity, reducing time-to-market, reducing costs, and aiding efficient development (MSC Software). Adams/Car provides a suite with numerous modules that allow analysis of all sorts of vehicles and vehicle subsystems.

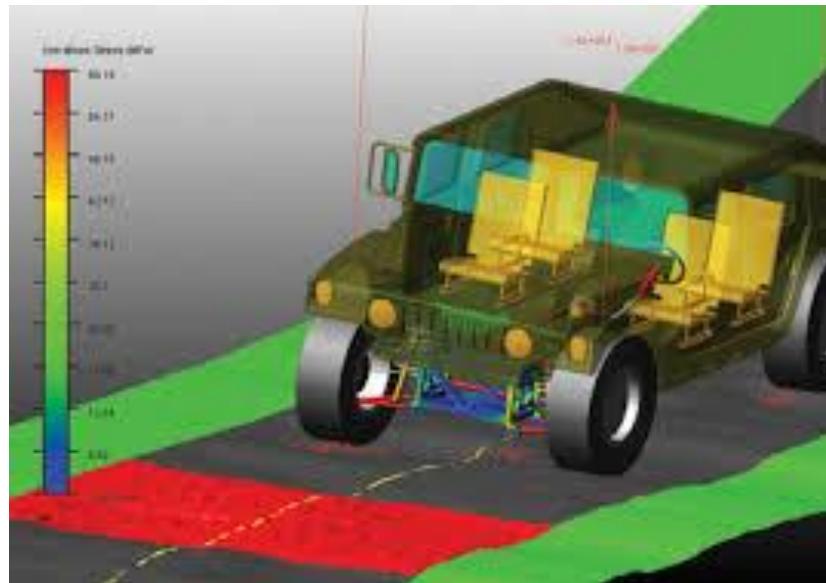


Figure 2. Automated Dynamics Analysis of Mechanical Systems (ADAMS/Car)

#### Screenshot

TORCS (The Open Racing Car Simulator) is an open-source high-fidelity 3D car racing simulator written in the C++ programming language (Wymann). It is available on Microsoft Windows, Mac OS X, and Linux and was designed to enable users to program an AI driver or controller as well as the ability to manually control a vehicle using a keyboard or joystick. This is a sophisticated racing simulator that captures many realistic and 3D dynamic effects that influence how real cars move. For instance, suspension coupling effects, motor gearing, and thin airfoil theory to determine the drag and downforce offered by the spoiler are all incorporated into this simulator.



Figure 3. The Open Racing Car Simulator (TORCS) Screenshot

CarSim is a high-fidelity 3D vehicle simulator that quickly and accurately simulates the dynamic behavior of vehicles and has become a standard in the automotive industry for investigating driver assistance technologies and vehicle dynamics (Snider, 2009). It is a commercially available simulator able to perform faster than real-time computations even on average computer hardware to simulate the dynamic behavior of a multitude of vehicles including, passenger cars, racecars, light trucks, and utility vehicles (CarSim). Over 800 variables are calculated while providing the ability to plot, analyze, and export the data to other software packages. CarSim also animates the simulated tests to provide a visual representation of each test.

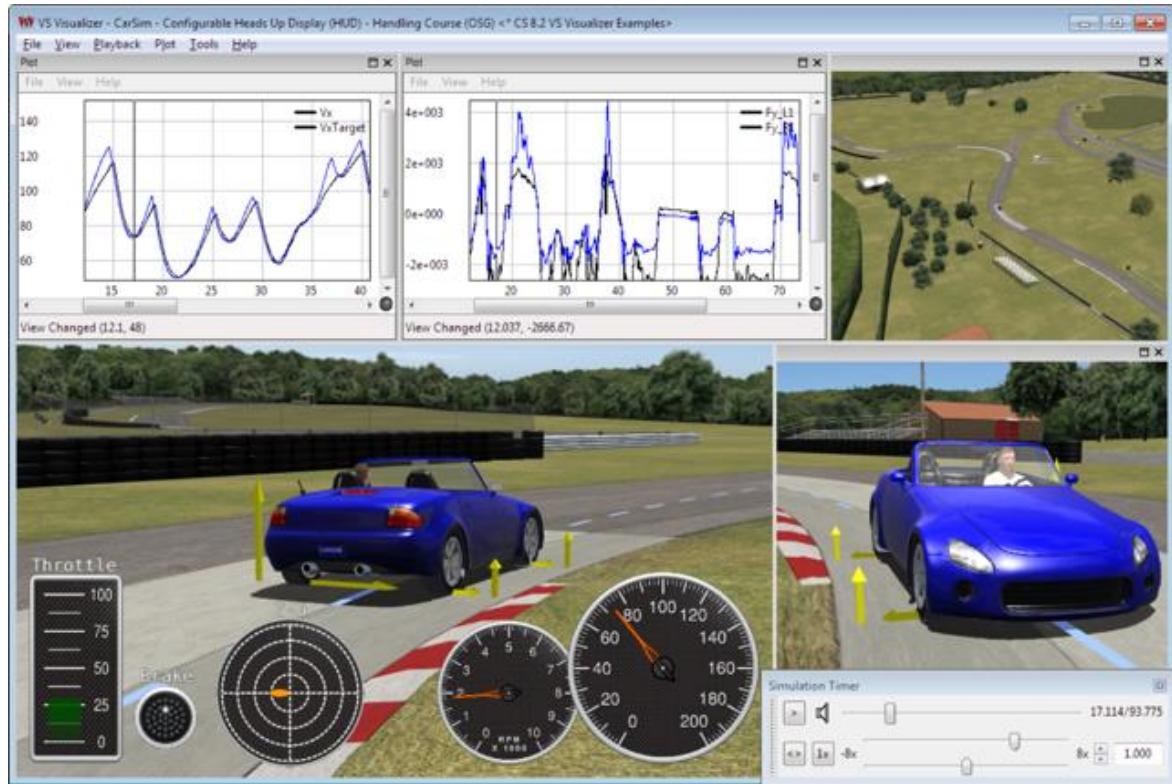


Figure 4. CarSim Software Screenshot

Prescan is a simulation platform equipped with a dedicated graphical user interface (GUI) pre-processor that allows users to build and modify traffic scenarios, infrastructure components, pedestrians, cars, weather conditions, and light sources for the development and validation of Advanced Driver Assistance systems (ADAS) (Tass International). Vehicle models can be equipped with different sensor types such as radar, laser, camera, ultrasonic, infrared, GPS, and antennas for vehicle communication. Control algorithms can be added and interfaced through Matlab/Simulink, which enables users to design and verify algorithms for data processing, sensor fusion, and vehicle controllers.

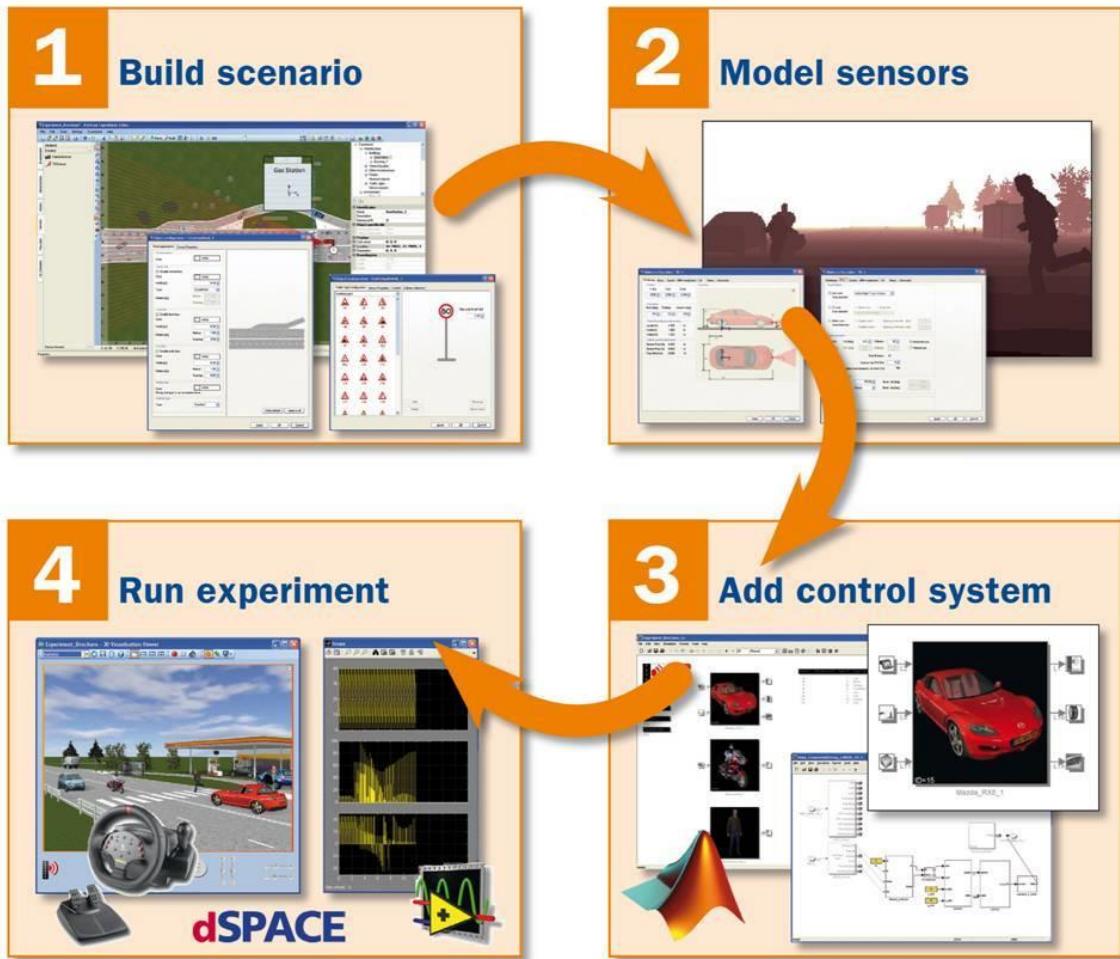


Figure 5. Prescan Simulation Software

There are some drawbacks using or modifying these high fidelity simulators in that they are extremely complex and may try to capture effects that have no merit in a particular area of research. For example, if using a simulator to develop autonomous algorithms then taking into account the fuel consumption and corresponding weight reduction effects may be of no use or may even slightly hinder development, especially if the vehicle you are attempting to model does not consume fuel but is rather powered by an electric motor. For a user new to these models and simulation software there is a large learning curve and the user may be forced to cope with certain effects in the model, such

as fuel consumption and gear changing, if there is no way to modify the underlying simulation software. In addition, many of these simulators do not take into account the hardware which runs the vehicle controller software which can have a huge impact on performance, stability, and latency. A simulation was created in order to aid in the development of a 1/10th scale model collision avoidance system with the intent to study human factors related to this technology. While simulation is a great tool when developing active and passive driver assistance technologies and autonomous algorithms it does not negate the need to run some actual ground tests at later stages of the design process; a human factors study yields more useful results if the driving is done in a real environment with actual hardware at risk as opposed to a simulated environment.

#### 1.4 Related Work and Current Technologies

As automation becomes more and more prevalent and commonplace in the transportation of people and goods, not only by ground vehicles, but also by air and sea, the relationship between people and machine becomes more and more crucial regarding safe operation of autonomous or semi-autonomous vehicles and their sub-systems. People's perception of how these systems are supposed to work, how they are operated, and their impact on society play a central role in determining the implementation, adoption, marketability, cost, and safety of such systems; this is especially true in order to achieve widespread implementation of such technologies. An analog of the growth, develop, and acceptance of autonomous controllers has occurred in the aerospace industry. Many incidents, both on land and in air, are caused by operator error, fatigue, distraction, and the operator's understanding and perception of how the autonomous or semi-autonomous system operates. Very few accidents involve mechanical, structural,

and/or electrical failures themselves. Thus, it is vital to not only study algorithms and safety systems that take control of the vehicle but also how users interact react and operate such systems and any pitfalls that may arise within this complex relationship between computer and operator.

Many new active and passive safety systems have come to market recently including Electronic Stability Control (ESC) and collision imminent braking systems in some Volvo and Mercedes-Benz models. Generally, these safety systems, especially those which interfere with driver commands, try to balance the response to keep the driver feeling in control. The minimal necessary interference principle seems to apply to these early adopters and will continue to do so for the foreseeable future. This is most likely because too much automation may enable drivers to become too distracted or careless and would increase liability for auto manufacturers. Although, fully autonomous vehicles are an area of active research and may well come to market in the near future. A few examples of such safety technologies are described below.

Another example of an active safety control system is Electronic Stability Control (ESC) or Electronic Stability Program (ESP). ESC minimizes the loss of control by detecting and reducing the loss of traction which improves vehicle stability (The Cooper Firm, 2014). When a loss of traction is detected, braking is automatically and independently applied to the wheels to prevent skidding and loss of control by minimizing the difference between the vehicle's heading and the desired heading. ESC is now mandated by law to be incorporated into new passenger vehicles in the US. The hope is that when active safety systems such as autonomous collision avoidance technology matures it will be similarly adopted and mandated. Since cars already have

onboard computers and various electronic controller units that control the vehicle, improve safety performance, and comfort the software used to run a collision avoidance controller could be incorporated without any additional hardware.

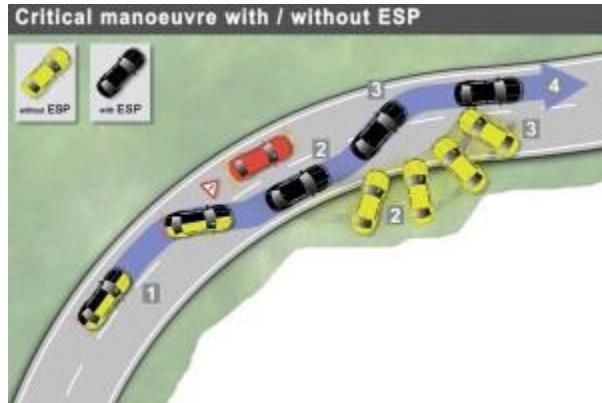


Figure 6. Electronic Stability Control (ESC)/Electronic Stability Program (ESP) Active Safety System

The PRE-SAFE braking system, developed by Daimler, provides an anticipatory braking system in the event of an impending collision (Daimler). The system utilizes a combination of long and short range radar sensors to detect approaching obstacles and prompts the driver to take action by means of visual and acoustic alerts. If the driver fails to act, the system assists with partial or emergency braking based on the estimated time-to-collision. The system initiates 40% braking 1.6 seconds before a detected impact and full braking at 0.6 seconds.



Figure 7. Mercedes Braking System

Google's fully autonomous self-driving car is a project that uses an array of robotic equipment and sensors to autonomously navigate and safely deliver passengers to their desired destination. The car is equipped with an array of sensors including: a Velodyne LiDAR sensor, GPS, multiple radar sensors, multiple cameras, and computers that constantly analyze the vast array of environmental data to find the optimal safe path to the target location. While this technology still has to overcome some regulatory issues, insurance questions, and consumer acceptance, it has logged over 700, 000 miles of autonomous driving and is aimed to be released publicly anywhere from 2017-2020 (Davies, 2014).

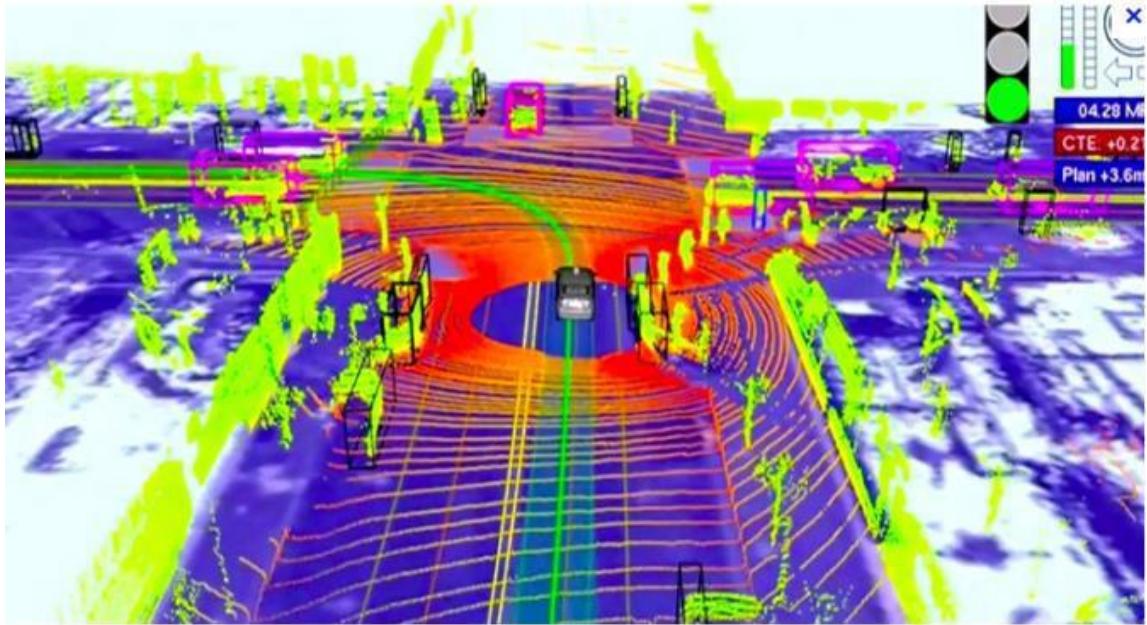


Figure 8. Google Self-Driving Car

### 1.5 Previous Work and Goals of This Study

Several Cal Poly teams have worked to create a steering based semi-autonomous collision avoidance system that in the case of an imminent collision takes control of the vehicle and autonomously steers around any obstacles. The system serves as a test bed for LiDAR based semi-autonomous collision avoidance research. This system features a LiDAR sensor used to dynamically detect obstacles within the environment. Various sensors and controllers are used including: wheel encoders to measure the speed of the vehicle, an inertial measurement unit (IMU) to provide measurements for a Model Predictive Controller (MPC) that provides rollover stability, a camera to provide the user with a driver's perspective view, and a custom microcontroller board to collect, relay, and actuate the response from a personal computer (PC) based control system. The hardware interfaces with a PC that runs the data processing and autonomous collision avoidance algorithms in a Matlab/Simulink model over a 100-foot Ethernet cable & a 100-foot USB

cable. A Rapidly Exploring Random Tree (RRT) algorithm is used for path-planning. The system outputs the steering and throttle commands necessary to safely maneuver the vehicle away from an impending collision. A threat metric is calculated to determine the threat of an imminent collision and when the vehicle controller should take control away from the driver. This process is repeated at a frequency of 20 Hz. The operator drives the 1/10<sup>th</sup> scaled vehicle via a joystick with standard steering wheel and pedal controls until the autonomous controller intervenes. The 2010 Crash Avoidance Team purchased the 1/10<sup>th</sup> scale Traxxas Slash RC car, installed wheel encoders, established a Matlab/Simulink Model to communicate with the onboard microcontroller hardware through a long Ethernet cable, and implemented & tested the RRT path-planning algorithm as detailed in (Miller, Ujiie, & Woods, 2011). Nikola Noxon's 2012 master's thesis installed an IMU and developed the MPC to provide rollover stability as described in (Noxon, 2012). The 2012 Crash Avoidance team implemented a LiDAR sensor for environmental perception, developed the segmentation and threat determination algorithms, and further refined the RRT path-planning algorithm according to (Stevens, Carlson, & Painter, Autonomous Collision Avoidance Final Design Report, 2013). The vehicle and closed-loop control system is shown below.

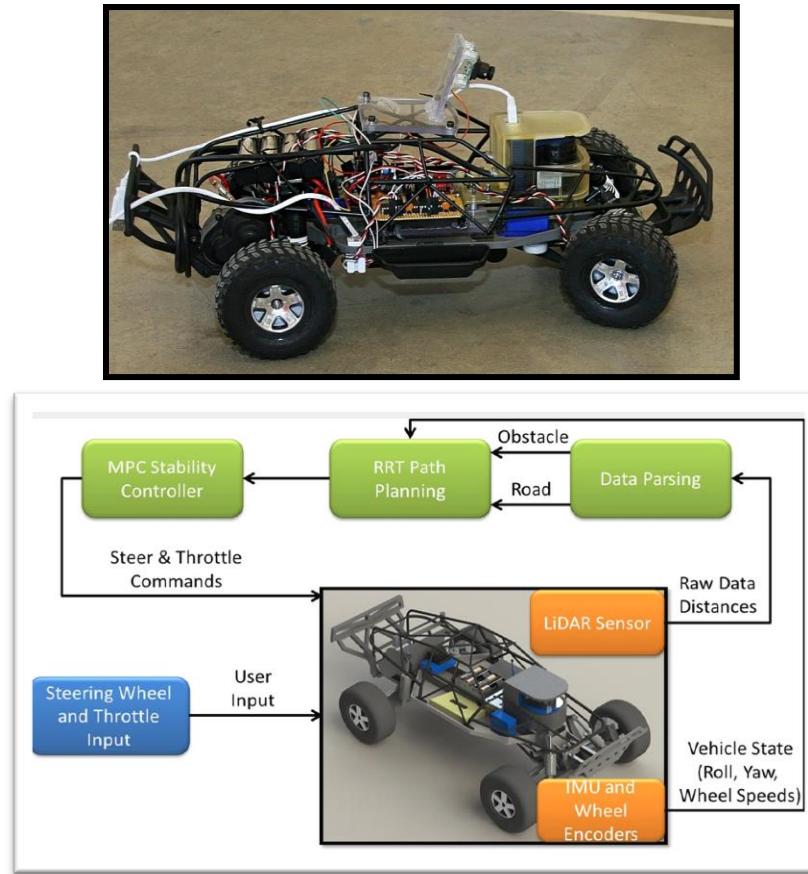


Figure 9. The 1/10th Scale Semi-Autonomous Vehicle Platform and Control Loop

This thesis aims to bridge the gap between simulation-based studies and empirical scaled vehicle testing to create a system that can be used in human factors studies. This entails bringing onboard the existing 1/10<sup>th</sup> scale collision avoidance system's software to an embedded single board computer (SBC) while maintaining the overall existing system architecture that features semi-autonomous collision avoidance and user control via a joystick with a driver's visual perspective. A SBC allows for a wireless platform that facilitates longer range tests where more aggressive maneuvers and complex obstacle behaviors can be studied than with the previous wired implementation. Ideally implemented, this platform will be used to provide insight into people's thoughts and feelings regarding collision avoidance technology. The SBC was selected in order to

operate the software at a sufficient rate while retaining enough capacity to allow for future development with the same hardware; a development board that offers a low learning curve and is host to a large community, support network, and project/documentation repository are primary considerations for this selection. The existing MATLAB/Simulink model based controller was replaced with (soft) real-time multithreaded C++ based software. The software is responsible for running the closed loop controller; this entails communicating with sensors, actuating hardware, and running the segmentation, threat determination, and path-planning algorithms while adhering to real-time constraints. This project will serve as a scaled research platform to study, improve, and optimize collision avoidance algorithms and components, but most importantly to help aid in evaluating people's perceptions and reactions to this new technology.

## 1.6 Overview of Remaining Chapters

This thesis is divided into six chapters. The first chapter discusses the trends regarding the current number of fatalities and injuries caused by car collisions in the United States. Chapter 1 introduces a few passive and active driver assistance technologies as well as some vehicle simulators. Finally, the previous work and goals of this study are explained. Chapter 2 details the theory related to vehicle dynamics, real-time vehicle controller development, computer simulation development & graphics, and communication & control protocols. Chapter 3 details the implementation of the hardware and software relating to the semi-autonomous collision avoidance system's user-control-station and the embedded controller used for scaled empirical testing. Chapter 4 details the development and features of the human-hardware-in-the-loop

simulator toolchain and its use as an aid in the development of the 1/10<sup>th</sup> scale semi-autonomous collision avoidance research platform. The testing results are shown and discussed in chapter 5 and conclusions and future recommendations are suggested in chapter 6.

## **Chapter 2      Theory**

This chapter describes the necessary topics that relate to this project's scope of work. The first section covers vehicle dynamics and the bicycle model as well as relevant material on evaluating the forces that appear in the model. The bicycle model is used in the hardware-in-the-loop simulator to produce the vehicle's dynamic behavior. The control laws and data processing algorithms incorporated into the controller's software are also described. Development of a real-time embedded system software architecture for embedded Linux is then covered. Next, the rendering and GUI development tools used in the hardware-in-the-loop simulation are described. The concept of pin multiplexing for single board computers is presented as well as the how to apply pin configurations to allow access to hardware peripherals. Pulse-width modulation is described in the context of controlling an RC vehicle's steer servo and motor. Finally, the UART hardware that facilitates the serial communication between the user control station and the vehicle is introduced.

### **2.1      Fundamentals of Vehicle Dynamics**

Vehicle dynamics, a branch of classical mechanics, encompass a broad range of modes of transportation – ships, airplanes, trains, and rubber-tired vehicles. Each of these vehicles is analyzed using the principles of dynamics. A dynamic analysis consists of identifying and quantifying all forces which affect these motions. The motions of a vehicle are accomplished by braking, accelerating, and cornering and are a response to the forces imposed on the vehicle. The driver may impose forces through a steering mechanism by changing the wheel's angle with respect to the current heading and by applying motor or brake torque to the wheels with the use of throttle and brake pedals,

respectively. Therefore, the control of the vehicle by the driver is dominated by the exchange of forces between the tire and the ground due to these control inputs; it is essential to understand and quantify these forces in order to study a vehicle's dynamic behavior. As Gillespie states in his book, "the primary forces by which a high-speed motor vehicle is controlled are developed in four patches-each the size of a man's hand-where the tires contact the road."

Newton's second law allows one to express the forces imposed on a vehicle dynamically as a system of differential equations (DE) or equations of motion that relate these forces, motions (position, velocity, and acceleration), and control inputs issued by the driver (steering, throttle, brake). These equations provide a predictive capability so that the necessary changes to reach a certain performance or design goal may be achieved, as Gillespie outlines. The differential equations are generated by developing a model of the vehicle's dynamic behavior and identifying the forces that cause its motion. These equations can be tailored to include (or exclude) forces that are relevant to the phenomenon of interest. This can be a very complex process and often the modeler simplifies this task by making assumptions and by including only the most pertinent components or forces to the vehicle's motion in order to adequately approximate reality; this step is vital in order to arrive at a manageable model that can be well understood and used to identify the influence of different vehicle's parameter. The complexity of the differential equations also directly affects the effort and time required to solve the equations for a solution of the motion of the vehicle. Considering the vehicle dynamics of a rubber-tired ground vehicle, many non-linear terms are exhibited by many components within the vehicle system, namely by the tires, which complicates its solution. Despite

forming a meaningful system of differential equations for the motion of a vehicle, it cannot be explicitly solved for an analytical closed-form solution unless certain assumptions are made; this can narrow the range of validity and limit the model's usefulness in analyzing all the possible motions of a vehicle, which handicaps the purely analytical approach for vehicle development. However, numerical integration allows the shortcomings of purely analytical methods to be overcome by approximation and can be readily implemented on personal computers in real-time. This allows for the analysis of more complete and complex vehicle systems with many components and subsystems that would prior have been virtually impossible to analyze. As pointed out by Gillespie, this also allows for simulation and evaluation of vehicles and their components before translation into actual hardware.

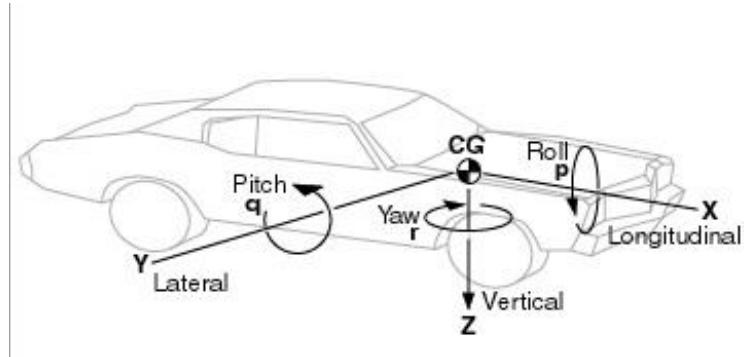


Figure 10. SAE Vehicle Axis System

Both fixed world coordinates and vehicle-fixed coordinate systems are used when studying vehicle dynamics. Transformation matrices can be used to go back-and-forth between these two coordinate systems. This is important as the sum of the forces and the corresponding vehicle accelerations are considered in the vehicle-fixed or SAE coordinate system. The SAE coordinate system, shown above in Figure 10, is a right-handed orthogonal coordinate system originating at the vehicle center of gravity (CG)

and is the frame of reference used to describe the forces on the vehicle. The relationship between the fixed world frame and the vehicle frame is given by the rotation matrix:

$$\mathbf{X} = \cos(\varphi)\mathbf{x} - \sin(\varphi)\mathbf{y} \quad (1)$$

$$\mathbf{Y} = \sin(\varphi)\mathbf{x} + \cos(\varphi)\mathbf{y} \quad (2)$$

This section introduced how a vehicle moves and how this motion may be described in terms of differential equations. The importance of tire dynamics on the overall vehicle's motion is emphasized. The concept of numerically integrating the governing differential equations on a computer sufficiently fast enough enables the determination of approximate solutions. The advent of numerical integration allows for the development of real-time vehicle dynamic simulations. Finally, the SAE coordinate system, used extensively while studying vehicle dynamics and its relation to a fixed world frame is shown. In the next section the bicycle model and its system of differential equations are introduced. The bicycle model and the numerical integration equations are described in detail in the remaining sections and form the basis for the hardware-in-the-loop vehicle dynamic simulator.

### 2.1.1 Bicycle Model

The analysis of the vehicle dynamics related to single-track bicycles are almost as old as the conception of the bicycle itself and have been used as a basis for simulation and controller design purposes for all types of vehicles. When used to analyze a four-wheeled, rubber-tired vehicle the typical bicycle model greatly simplifies the vehicle's lateral and longitudinal dynamics. The front and rear tires are collapsed together that reduce the four-wheeled vehicle to a single-track vehicle. By applying simplifying assumptions, linearized differential equations with closed form solutions can be found.

While the simplifications applied to the bicycle model do make solutions easier and faster to obtain, there is a tradeoff in realism such as the lack of lateral load transfer due to the single-track configuration of the model. However, the consideration of lateral load transfer is not of crucial importance to vehicle controller development and the simulation of vehicle dynamics. Developing a model that balances the solution and realism is an important aspect of studying vehicle dynamics. Previous research has confirmed that the classic bicycle model accurately describes the dynamics of actual vehicles in situations where the vehicle experiences less than 0.3 g's. This also corresponds to the linear response regime associated with normal driving with lateral accelerations up to 0.4 g's as outlined in the classic set of papers by Milliken, Segel, and Whitcomb in 1956 and 1957 (Dixon, 1996). With the above assumptions, closed form solutions can be found for the bicycle model. While explicit analytical solutions are useful for steady-state maneuvering, they pose problems when analyzing and developing a simulation where the vehicle can accelerate from rest and operate under a wide dynamic range

In order to model the dynamic behavior of a vehicle's motion each of the vehicle's components are considered in order to determine their dynamic contribution to the forces imposed on the vehicle. One way to simplify the analysis is by considering only the components that exert the dominant forces most pertinent to the vehicle's motion, especially for preliminary models. The components most pertinent to a vehicle's motions are the tires, motor, aerodynamics, and friction as outlined by Dixon. The vehicle body is considered for its contribution to inertia, the motor is considered for production of acceleration forces that affects the capability of the tires to produce cornering forces, and the aerodynamic drag & rolling resistance is considered as the

resistive elements that impede the vehicle's motions. Each component's force contribution must be modeled so that its effect may be incorporated into the overall governing differential equation. Shown below, Figure 11 is a diagram of the dynamic bicycle model and tire forces.

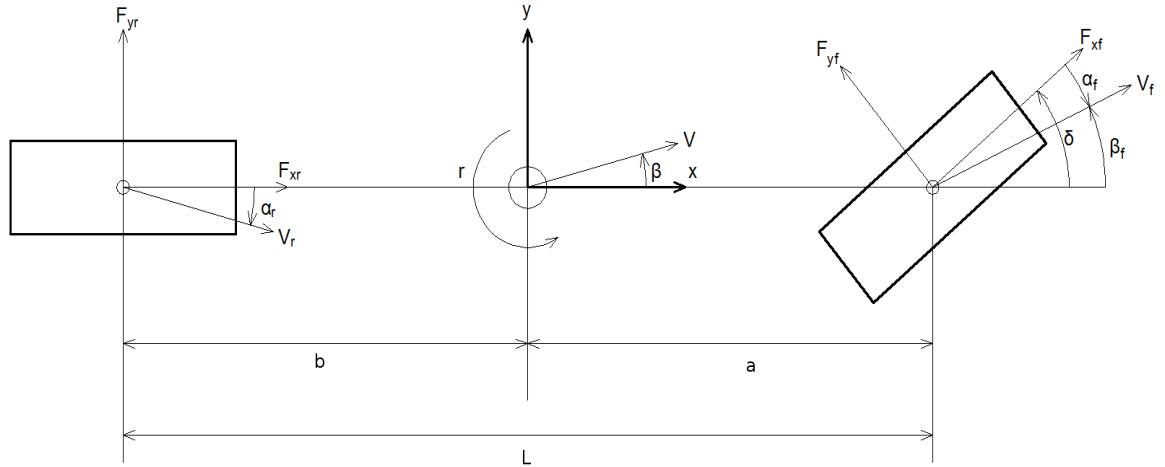


Figure 11. The Dynamic Bicycle Model

Applying the summation of the forces according to Newton's 2<sup>nd</sup> Law to Figure 11 yields the following equations for the dynamic three degree of freedom model (Gillespie (Michielsen, 2013) (Milliken & Milliken, 1995) (Dixon, 1996) (Monster, 2003)):

$$\sum \mathbf{F}_x = F_{xf} \cos(\delta) + F_{xr} - F_{yr} \sin(\delta) = m(\dot{V}_x - \dot{\phi}V_y) \quad (3)$$

$$\sum \mathbf{F}_y = F_{yf} \cos(\delta) + F_{yr} + F_{xf} \sin(\delta) = m(\dot{V}_y + \dot{\phi}V_x) \quad (4)$$

$$\sum \mathbf{M}_z = a(F_{yf} \cos(\delta) + F_{xf} \sin(\delta)) - bF_{yr} = I_z \ddot{\phi} \quad (5)$$

Rearranging the above equations, we obtain the following:

$$\dot{V}_x = \frac{F_{xf}}{m} \cos(\delta) + \frac{F_{xr}}{m} - \frac{F_{yr}}{m} \sin(\delta) + \dot{\phi}V_y \quad (6)$$

$$\dot{V}_y = \frac{F_{yf}}{m} \cos(\delta) + \frac{F_{yr}}{m} + \frac{F_{xf}}{m} \sin(\delta) - \dot{\phi}V_x \quad (7)$$

$$\ddot{\phi} = \frac{a}{I_z} (F_{yf} \cos(\delta) + F_{xf} \sin(\delta)) - \frac{b}{I_z} F_{yr} \quad (8)$$

The above system of coupled differential equations that make up the bicycle model relates the longitudinal, lateral, and angular accelerations to the forces and moments on the vehicle, respectively. These equations can be numerically integrated to solve for the longitudinal, lateral, and angular velocities and positions of the vehicle at each simulation time step. Equation 6 relates the longitudinal acceleration to the longitudinal forces on the vehicle. The forces are composed of components of the lateral and longitudinal forces on the front steerable wheel, the tractive force on the rear wheel, and the yaw rate and lateral velocity. Equation 7 relates the lateral acceleration to the lateral forces on the vehicle, which consists of components of the forces on the front wheel, the lateral force on the rear wheel, and the yaw rate and longitudinal velocity. Equation 8 relates the angular acceleration to the imposed moments, which are composed of the forces on the front wheel and the lateral force of the rear wheel and the moment arms. The assumptions typically used with the bicycle model involve linearizing the equations about a steady state longitudinal velocity and applying the small angle approximation. Even with these assumptions, the model is very useful and widely studied in regards to vehicle dynamics and controller design. The small angle approximation of the steered wheels allows the lateral forces to be expressed as linear functions of the slip angle. The constant velocity assumption typically used in the bicycle model decouples the lateral acceleration equation (Equation 7) from the longitudinal velocity as the longitudinal acceleration (Equation 6) becomes zero due to the constant longitudinal velocity assumption. These assumptions result in a two degree of freedom model. To better suit the needs of this research, the typical bicycle model's common simplifying

assumptions were not applied to preserve the usefulness of the model throughout the dynamic range of the vehicle. This also preserves the coupling of the longitudinal and lateral accelerations to the lateral and longitudinal velocities and provides a three degree of freedom model. Three degrees of freedom allow the simulated vehicle to rotate and translate in a two dimensional plane while neglecting pitch and roll. Also introduced into the model for this project was aerodynamic drag and rolling resistance that are typically ignored with the classic bicycle model. This provides a more general model with more tuning parameters that may be used to better match the simulation to reality. Altering the longitudinal dynamic Equation 6 to include aerodynamic drag and rolling resistance leads to the equation shown below:

$$\dot{V}_x = \frac{F_{xf}}{m} \cos(\delta) + \frac{F_{xr}}{m} - \frac{F_{yf}}{m} \sin(\delta) - F_{drag} - R_{xf} - R_{xr} + \dot{\phi} V_y \quad (9)$$

This section established the bicycle model and the equations that are numerically integrated within the simulation to model the vehicle's dynamics. With the bicycle model established, each of the forces appearing within it needs to be evaluated. In the remainder of this section, the equations to determine the forces provided by the motor, supported by the tires, and resulting from friction are described.

### 2.1.2 Tire Models and Lateral Forces

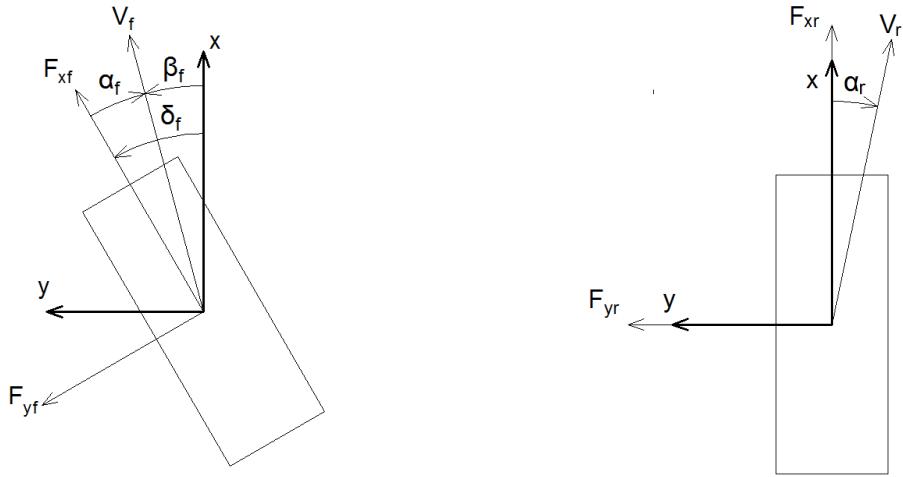


Figure 12. Forces Acting on the (Steerable) Front and Rear Tire

As Dixon points out, the essential function of a tire is to interact with the road in order to produce the forces necessary to support and move the vehicle by accelerating, braking, and cornering. The forces produced by the tires depend on many factors such as slip angle, slip ratio, inflation pressure, temperature, size, geometry, tread design, load, contact patch, etc. Detailed by Milliken, the tires are the primary source of the forces that provide the control and stability of the vehicle. Within the tire's contact patch the tire sticks to the road surface as it rolls and deforms to provide the frictional and elastic forces necessary for accelerating, braking, and cornering. Gillespie discusses the mechanisms responsible for the tire-road friction within the contact patch such as the surface adhesion between the rubber and the road surface as well as the bulk hysteretic friction energy loss as the tire deforms. The friction that generates the forces required to maneuver the vehicle depend on some slip occurring at the tire road interface. Both the longitudinal and lateral tire forces exhibit an elastic distortion region and a sliding or frictional region as shown and labeled in Figure 13.

P215/60 R15 Goodyear Eagle GT-S (shaved for racing) 31 psi.

For a given load, in this case 1800 lb.

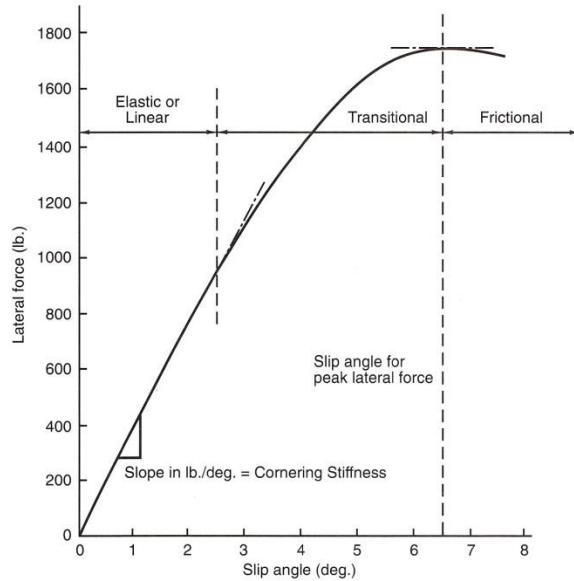


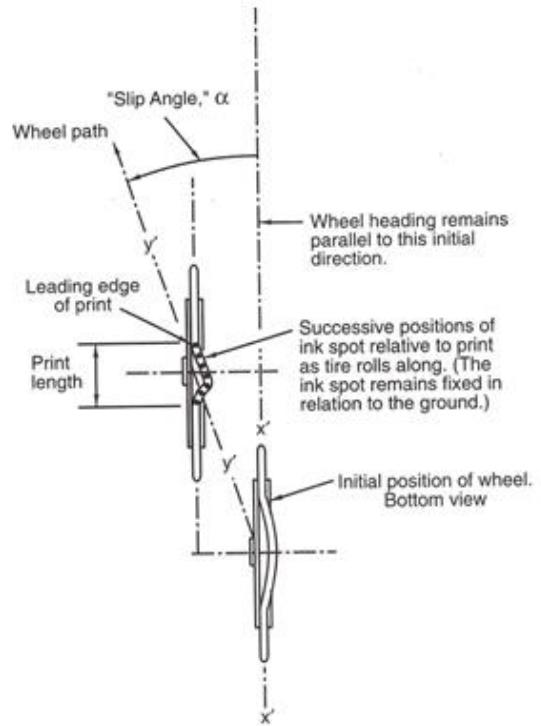
Figure 13. Lateral force versus slip angle

The longitudinal force developed in the tire footprint allows the vehicle to change its speed by accelerating and braking. Due to the stretching and adhesion of the tire in the contact patch, the wheel will exhibit a difference in the driven (or braked) angular velocity compared with a free-rolling wheel at a ratio called the slip ratio. This means that the driven rear wheels will be rotating slightly faster than the non-driven front wheels, as expressed in Equation 10. For a free rolling wheel the slip ratio is zero, for a fully locked wheel the slip ratio is -1, and for an accelerating wheel the slip ratio is positive. The force that accelerates the car depends on the amount of wheel slip or slip ratio.

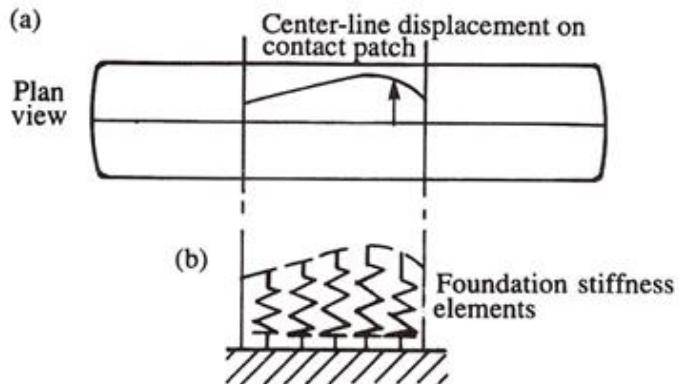
$$SR = \frac{\omega_R}{V} - 1 \quad (10)$$

When a vehicle is cornering at high speeds, the wheels are no longer moving in the direction they are pointed and the tires develop lateral cornering forces. When a tire is subjected a lateral force, it is pushed sideways as the rubber deflects and the tire tends to

move a small amount in the direction of the subjected force at an angle called the slip angle. A greater lateral force causes greater deflection until the tire begins to slide and the lateral force generated by the tire roughly levels off. The front and rear lateral forces depend on the slip angle between the tire's heading and its direction of travel. The slip angle is defined as the angle between a rolling wheel's direction of travel and the wheel's heading which may be calculated from the ratio of the lateral velocity and the longitudinal velocity of the wheel. Globally, this causes the vehicle to point in one direction and travel in another direction that differs by the global slip angle. These angles are labeled in Figure 14. Many tire models exist in order to model the forces and the parameters that influence them; both empirical and analytical models are widely used. Analytical models tend to be used to form a basic understanding of tire dynamics and empirical models are more widely used to model the overall tire behavior.



(Milliken & Milliken, 1995)



(Dixon, 1996)

Figure 14. Mechanism of tire lateral force in elastic range

One analytical tire model is the elastic model that is composed of many small elastic elements that may be considered to act independently. This model is shown in

Figure 14. If forced by the ground, each element resists with a given stiffness by displacing from its null position. Other models used to understand the principals of tire forces and deflections are the beam model and the string model which may be found in the literature, but none represent the true complexities of real tires. Although the elastic model gives an intuitive sense of how a tire generates the lateral forces required for cornering, most models, especially those that are computationally efficient, are empirically based and forgo any attempts to derive tire behavior from material properties and tire construction.

The local sideslip, or slip, angles at the front and rear tire are:

$$\alpha_f = \beta_f - \delta \quad (11)$$

$$\alpha_r = \beta_r \quad (12)$$

The global sideslip angle for each tire is defined as

$$\beta_i = \tan^{-1} \left( \frac{V_{yi}}{V_{xi}} \right) \text{ where } i = f, r \quad (13)$$

According to the relative velocity equation:

$$V_i = V + \dot{\phi} \times r_i \quad (14)$$

Combining Equations 11-14, the local sideslip angles of the front and rear tires are obtained:

$$\alpha_f = \tan^{-1} \left( \frac{V_y + a\phi}{V_x} \right) - \delta \quad (15)$$

$$\alpha_r = \tan^{-1} \left( \frac{V_y - b\phi}{V_x} \right) \quad (16)$$

Detailed by Milliken, the lateral forces originate where the tires contact the ground and generate the forces responsible for turning and accelerating the vehicle. As described above, the tire operates within three regions, shown in Figure 13, called the

elastic, transitional, and frictional regions. To accurately model a tire's lateral force the empirical model used must exude the characteristics of the curve. One widely used empirical lateral force tire model developed by Pacejka is known as the *Magic Formula*. This formula is used extensively in industry, in simulators, and is incorporated into the bicycle model used in this project to capture non-linear effects that may be encountered at high speeds where the vehicle controller may operate. This enables some of the tire's complex behavior to be captured such as force saturation, different road surfaces, and the lateral force dependence on the load. Although, this model doesn't capture some other interesting dynamic effects such as tire force lag or the time it takes for the tire to develop the lateral force for a given amount of applied slip angle. The *Magic Formula* is given below in Equation 17.

$$F_{yi} = D \sin\{C \tan^{-1}[B\alpha_i - E(B\alpha_i - \tan^{-1}(B\alpha_i))]\} \quad (17)$$

where  $i = f, r$

$$D = \mu F_{zi}$$

$$\text{and } B = \frac{c_{ai}}{CD}$$

The peak factor (D) controls the curve's peak lateral force which depends on the coefficient of friction between the tire and the road and the vertical load on the wheel. The stiffness factor (B) controls the elastic linear region of the curve's lateral force development. The shape factor (C) and the curvature factor (E) control the curve's shape and curvature respectively. A typical curve given by Equation 17 is shown below in Figure 15.

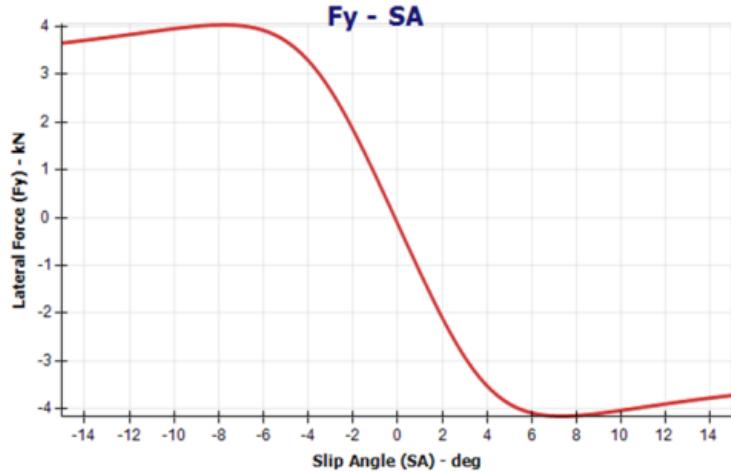


Figure 15. Example Magic Formula curve

Since a tire must generate both the lateral and longitudinal forces required for cornering and accelerating/decelerating they both compete for the tire's grip. A tire may only support so much force before it loses grip and slides. The limit of a tire's grip and the interplay between providing both lateral and longitudinal forces can be represented with a friction circle, traction circle, circle of forces, or friction ellipse diagram. An example is shown in Figure 16. This diagram is a graphical representation of the friction limits and forces on the tire and provides a convenient way to consider the competition that braking, accelerating, and cornering are in for tire grip. The friction limit is determined by the vertical load and the coefficient of friction between the tire and road surface; the total force on the tire may not exceed this limit without sliding. In reality, the friction circle is closer to an ellipse than a circle due to the difference in the lateral and longitudinal coefficient of friction. The limit of a tire's grip is strongly speed dependent as is the coefficient of friction between the tire and the road-surface. This has important implications such as, the tire's grip limit changes while the vehicle is in motion. There are four commonly used friction speed models as discussed by Dixon: constant friction

coefficient  $\mu$ , static and dynamic friction coefficients  $\mu_s$  and  $\mu_d$ , and speed dependent friction coefficients.

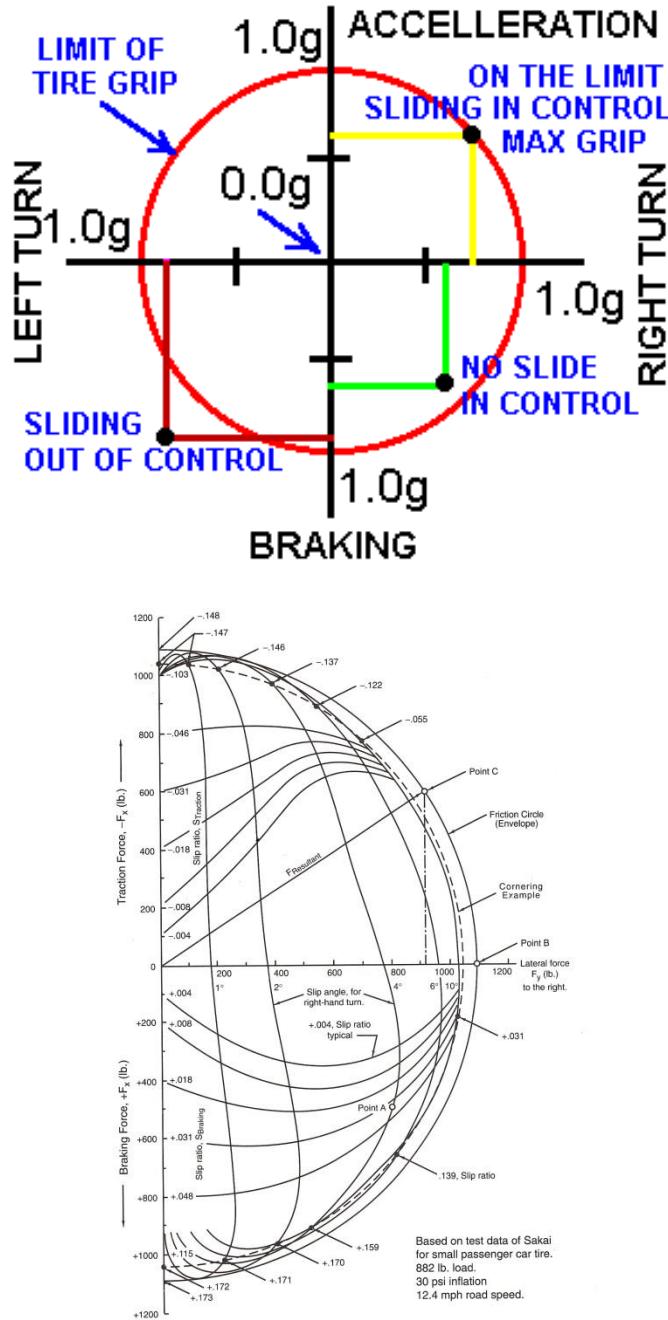


Figure 16. Circle of forces, traction circle, friction circle, or friction ellipse concept

As Gillespie states, the early study of vehicle dynamics, especially related to steering and cornering, was greatly hampered by the lack of knowledge of tire mechanics.

However, this changed in 1931 when a tire dynamometer was built. This machine could measure the mechanical properties of the pneumatic tire and offered a much deeper understanding of vehicle dynamics (Steele, 1955). This allowed engineers to develop mechanistic explanations of turning behavior and laid the framework for a majority of what is understood today regarding vehicle dynamics. The understanding of the forces generated by pneumatic rubber tires are a crucial element to understanding ground vehicle dynamics. Incorporation of the *Magic Formula* tire model allows simulation of lateral skidding and sliding effects. Due to the empirical nature of the *Magic Formula*, experiments must be performed in order to evaluate the parameters used in this tire model. These experiments are commonly performed with a tire dynamometer. To accommodate obtaining such data, several testing facilities with highly specialized tire testing machines collect large amounts of data; The Calspan Testing Facility is shown below in Figure 17. A scaled tire testing machine utilizing a treadmill was developed by Brennan as shown in Figure 18. A similar cornering stiffness testing machine was built for this project to estimate the parameters for the *Magic Formula* and is discussed in a later chapter. The data points obtained for a given vertical load and road surface typically fall on a curve similar to that seen in Figure 13 and the different ranges of tire operation (elastic, transitional, and frictional) can be seen.

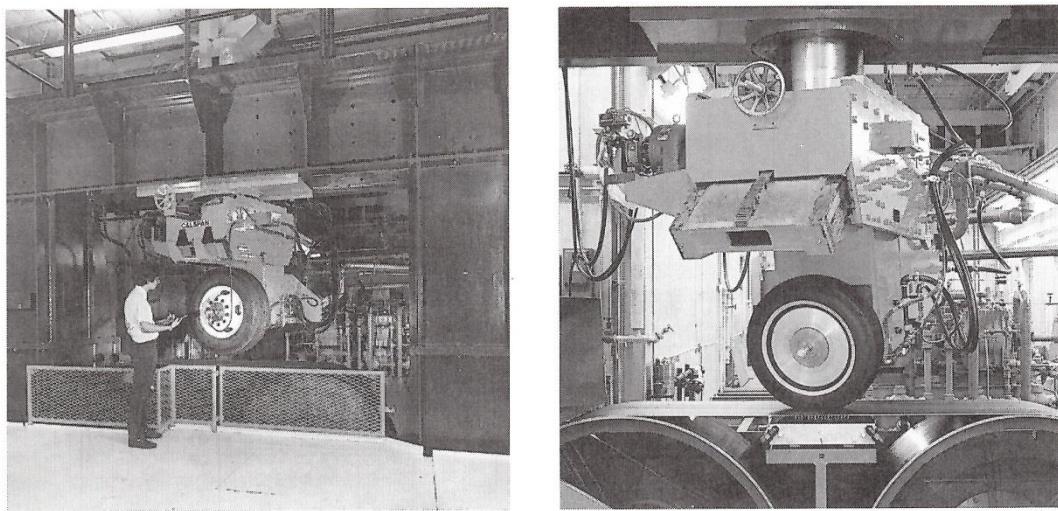


Figure 17. Calspan Tire Research Facility, TIRF

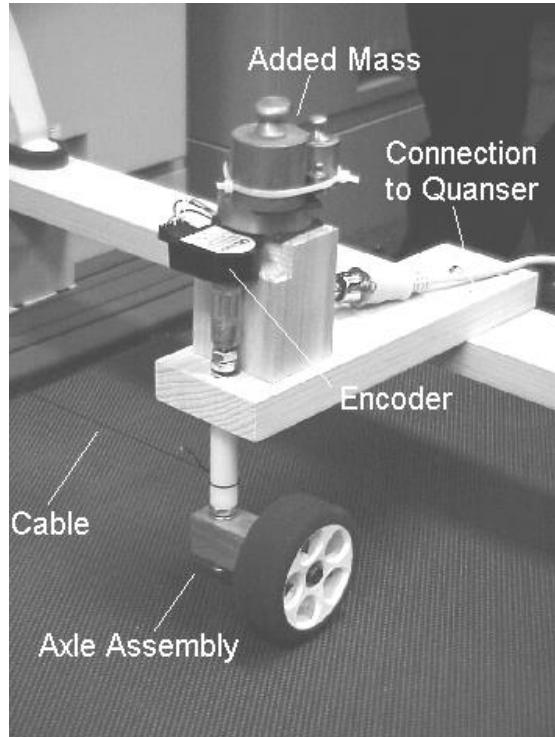


Figure 18. Cornering Stiffness Testing Apparatus

This section introduced the mechanisms in which tires generate forces and the characteristics of tire operation. An analytical elastic tire model was introduced to show how the concept of a slip angle was conceived. The *Magic Formula* tire model was described to provide a means to empirically describe a tire's performance and to enable

evaluation of the tire's lateral force contribution as a function of the tire's slip angle. Evaluation of the tire slip angles using the above equations allows the lateral forces generated by each tire to be evaluated and used in the bicycle model during simulation. Determining the slip angles and lateral forces is essential to dynamic simulation of a vehicle that is cornering. Next, the friction circle was shown as a way to consider the interplay between the tire's lateral and longitudinal forces with respect to the available force and the tire's traction limits. Finally, a tire dynamometer machine was shown as a method to measure the tire forces as a function of the slip angle which enables evaluation of the *Magic Formula* tire model's parameters.

### 2.1.3 Engine and Drivetrain Modeling and Power Limited Acceleration

Modern day vehicle engines and drivetrain assemblies are extremely complex electromechanical systems that require simplification during model development in order to make their simulation practical. Modeling an engine through its arbitrary components such as cylinder, valves, cams, and flywheel becomes unreasonable and loses a generic and modular way to describe an engine's performance. In order to incorporate power-limited acceleration into the analysis of an accelerating vehicle the engine characteristics must be examined. As Gillespie's states, "the performance characteristics of engines may be characterized by their torque and power curves as a function of speed." Some typical gasoline engine power curves are shown below.

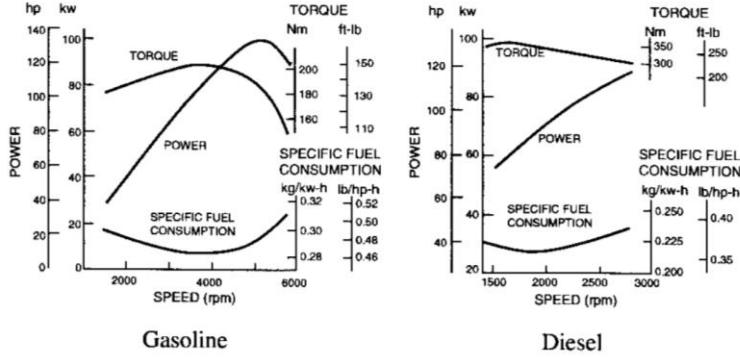


Figure 19. Performance characteristics of gasoline and diesel engines

A simple linear model describing a DC motor is adopted as a primitive approach to model the torque and power curve as a function of speed. While this is not necessarily the most accurate way to model a brushless three-phase motor, it provides a simple modular, intuitive, and generic motor model. The linear DC motor model can be written as (Nise, 2011):

$$T_m = \frac{-K_b K_t}{R_a} \omega_m + \frac{K_t}{R_a} E_a \quad (18)$$

$$\text{Where } K_b = \frac{1}{K_V}$$

In the linear motor model above the motor torque equals the armature voltage divided by the armature resistance and multiplied by the motor torque constant that decreases linearly with increasing angular velocity. The tractive force at the drive wheel(s) is found by considering the motor torque, effective power-train gearing ratio, and vehicle tire radius according to the equation below.

$$F_{x,i} = \frac{T_m G}{R} \quad \text{where } i = f, r \quad (19)$$

Assuming the engine can provide the power, the acceleration may be limited due to a lack of traction that is dictated by the normal load on the tire and the coefficient of friction

between the tire and the road surface. In this case the traction force is limited as shown in the equations below and represents the friction circle envelope:

$$F_{x,i \text{ limit}} = \mu W_i \quad \text{where } i = f, r \quad (20)$$

As shown in Dixon, a speed dependent friction coefficient can be included to improve the friction model and capture the friction coefficient's speed dependence. A speed dependent dynamic friction coefficient was included and may be calculated as follows:

$$\mu_D = \mu_S(1 - KV) \quad \text{where } K=0.012 \text{ s/m} \quad (21)$$

This section introduced a simple linear motor model to simulate the motor that accelerates the vehicle. The equations presented allow consideration of limited acceleration due to a lack of power or traction. Finally, the dynamic coefficient model used was described in the context of determining the friction circle envelope.

#### 2.1.4 Static Loads and Longitudinal Load Transfer on Level Ground

Since the center of gravity of the vehicle does not lie exactly in the geometric center the front and rear vertical loads vary. This is an important consideration as the lateral forces and tire grip or friction circle envelope for each tire are functions of the vertical load it is subjected too. As discussed in the literature, the following equations describe the static loads on the front and rear tire of a vehicle.

$$W_{fs} = W \frac{a}{L} \quad (22)$$

$$W_{rs} = W \frac{b}{L} \quad (23)$$

Confusingly, weight transfer and load transfer describe two distinct effects. Weight transfer typically refers to the change in the center of mass as the vehicle accelerates due to suspension compliance and cargo movement. Load transfer customarily refers to the change in the load the different wheels support during

acceleration. For a single-track bicycle model with a rigid frame, only the longitudinal load transfer is present as no change in the center of mass occurs. Alternatively, load transfer can have significant impact on an accelerating/decelerating vehicle as the load transfer can affect a tire's ability to grip the road surface and to produce lateral cornering forces. In addition, load transfer effects may be experienced by vehicles with no suspension at all as no change in the center of mass is required. The dominant forces that accelerate the vehicle occur in the tires' contact patches and do not align with the vehicle's center of mass; this causes the generation of moments. Generally, the weight transfer effect is far less important to a vehicle's dynamic behavior than the load transfer. When the vehicle is accelerating on level-ground, load is transferred longitudinally from the front to the rear axle in proportion to the amount of acceleration. The mathematical description is shown in the equations below.

$$W_f = W \left( \frac{a}{L} - \frac{\dot{v}_x h}{g L} \right) = W_{fs} - W \frac{\dot{v}_x h}{g L} \quad (24)$$

$$W_r = W \left( \frac{b}{L} + \frac{\dot{v}_x h}{g L} \right) = W_{rs} + W \frac{\dot{v}_x h}{g L} \quad (25)$$

### 2.1.5 Rolling Resistance

The rolling resistance is associated with the effect of the energy lost while the tires constantly deform while rolling as well as the frictional loses due to adhesion. This force always opposes the motion of the vehicle. As Dixon points out, the total rolling resistance is apportioned typically as 90% material hysteresis, 8% surface friction, and 2% air friction. Usually, rolling resistance coefficients ranges from 0.01 to 0.025 for cars. However, the energy lost during the deformation of the tire as it rolls is several times higher than other resistances. At low-speeds the rolling resistance is the dominant force that resists vehicle motion. This effect is included in this project as the vehicle that is

used and simulated may experience significant resistive forces that slow it down and bring it to rest when no throttle is applied. By including this effect, the simulated vehicle's dynamic behavior may be more easily matched at low speeds with no throttle input; the inclusion provides another tuning parameter to better match the simulation with reality. In addition, this effect can be completely neglected by selecting a rolling resistance coefficient of zero. As shown in Gillespie, the rolling resistance forces are:

$$\mathbf{R}_{xf} + \mathbf{R}_{xr} = \mathbf{f}_r (\mathbf{W}_f + \mathbf{W}_r) = \mathbf{f}_r \mathbf{W} \quad (26)$$

### 2.1.6 Aerodynamic Drag

As the result of the air flow over the vehicle, forces and moments are imposed on the vehicle due to the vehicle's skin friction. At high speeds this force becomes the dominant resistive force and is proportional to the vehicle's velocity squared. Due to the complexity of the flow over a vehicle, semi-empirical models are used to represent this effect. This effect is included in this project as it largely determines the road load power the motor must overcome to accelerate the vehicle. Inclusion of drag also provides a high speed tuning parameter to better match the top speed where the motive power becomes equal to the road load power. This effect is crucial in order to incorporate power-limited acceleration into the simulation but may also be neglected by selecting a drag coefficient of zero if desired.

$$\mathbf{F}_{drag} = \frac{1}{2} \rho C_D A |V| V \quad (27)$$

### 2.1.7 Low-Speed versus High-Speed Model

When considering cornering there exists two regimes of interest: low-speed and high-speed cornering. Cornering at low speeds, such as in parking lots, a vehicle moves along a curvilinear path and experiences very small lateral acceleration; the lateral forces

are essentially negligible. The bicycle model estimates the cornering forces produced by the tires based on the slip angles calculated with Equations 15-16. When attempting to use these equations at very low speeds the solutions quickly diverge to infinity because the vehicle's longitudinal velocity appears in the denominator. Simulation instability appears when these equations are used in simulation at low speeds that consequently leads to extremely unrealistic vehicle behavior. This poses a significant problem when developing a simulation to model the dynamics of a vehicle which may operate in all dynamic regions. The classic bicycle model assumes a constant longitudinal speed and thus avoids this problem. A kinematic bicycle model is usually sufficient to avoid this. A geometry based approach is taken at low speeds by switching to a kinematic bicycle model to retain realistic vehicle motion. Both a low-speed model and high-speed model are incorporated into the simulation to allow the simulator to model low-speeds where the lateral tires forces produced are negligible and to model high-speed slip phenomenon. The term low and high speed are relative.

#### 2.1.8 Numerical Integration

In order to advance the simulation forward in time, the dynamic vehicle model must also be propagated in time to approximate the dynamic response by solving the governing differential equations at each time step. This involves numerical integration of the governing differential equations in order to calculate the vehicle's position, velocity, and acceleration. With personal computers, there no longer exists a need to rely on simplified vehicle dynamics models and the full dynamic vehicle behavior may be simulated. Many integration schemes exist with varying accuracy and computational effort. A simple forward Euler Method, shown below, was used to numerically integrate

each differential equation (Milgram, 2003). This method allows snapshots to be taken in time by using the current snapshot to numerically estimate the future snapshot and permits estimation of the nonlinear and mutually dependent governing differential equations.

$$\mathbf{y}_{i+1} \approx \mathbf{y}_i + \mathbf{f}(\mathbf{x}_i, \mathbf{y}_i)(\mathbf{x}_{i+1} - \mathbf{x}_i) \quad (28)$$

One problem associated with numerical integrators is there potential to become numerically unstable and ‘blow up.’ This occurs if the time step between updates, or snapshots, is not small enough and the values will quickly jump to infinity as small errors are multiplied quickly into larger and larger errors. Eventually, these errors will propagate into all coupled equations degrading the simulator’s ability to realistically estimate the vehicle’s motion. Decreasing the time step can help but increases the required central processing unit (CPU) cycles. With greater computational expense, an alternative and more robust numerical integration algorithm could be implemented such as the fourth-order Runge-Kutta method (RK4).

### 2.1.9 Steering Methods for Path-Tracking

In order to steer an autonomous or semi-autonomous vehicle along a desired path a control law must be developed. A very popular path tracking method used in robotics exploits the geometric relationship between the vehicle and the path. This path tracking method evolved from the geometric condition required for steering to maintain the wheels perpendicular to their arc of motion as first investigated by Georges Langensperger in 1816 and described by Dixon. This arrangement was found to be highly desirable as it minimizes the friction at the wheels during low-speed maneuvering. The importance of this invention was recognized by Ackermann who, with agreement with

Lagensperger, took out a patent and is now commonly known today as Ackermann steering. As described by Snider, this results in a simple geometric relationship described below (Snider, 2009).

$$\tan(\delta) = \frac{L}{R} \quad (29)$$

The pure-pursuit path tracking method is a common simplification of Ackermann steering which is used in conjunction with the bicycle model to form a control law for path-tracking. The pure-pursuit control law can be derived from the pure-pursuit geometry shown in Figure 20 and can be written as:

$$\delta(t) = \tan^{-1} \left( \frac{2L \sin(\alpha(t))}{l_d} \right) \quad (30)$$

The characteristics of the pure-pursuit tuning are summarized as: decreasing the look-ahead distance results in higher-precision tracking and eventually oscillation while increasing the look-ahead distance results in lower precision tracking with greater stability. This becomes important in this research when tuning the controller to follow the path generated by the path-planning algorithm which makes exclusive use of this control law. While the pure-pursuit path-tracking control law is an imperfect solution it can perform quite well in common driving scenarios and geometric path trackers are simple to understand, implement, and pose very little computational overhead.

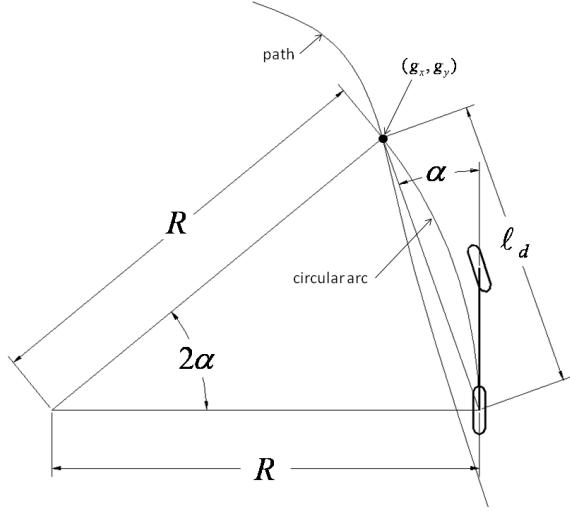


Figure 20. Pure-Pursuit Geometry

#### 2.1.10 Segmentation and Threat Determination

The data from the environmental sensors must be processed and analyzed in order for the path-planning algorithm that finds a safe path which the vehicle attempts to follow. The point-cloud from the LiDAR sensor is segmented to determine obstacles surrounding the vehicle and to identify the road boundaries. This segmentation occurs in three steps: point clustering, parsing of point clusters to obtain any obstacle's geometric properties, and obstacle tracking and velocity estimation. By clustering adjacent points together, discrete sets of ordered points are created that form each detected obstacle's definition. To evaluate the geometric properties of these obstacle definitions and to generate the bounding boxes around the detected obstacles, the clustered points making up each obstacle definition are parsed using a method based on the Ramer-Douglas-Pecker algorithm and outlined in (Stevens, Carlson, & Painter, Autonomous Collision Avoidance Final Design Report, 2013). The size and location of the bounding boxes around each detected obstacle can be controlled via various algorithm parameters. For each detected obstacle an associated threat parameter is calculated. If the detected

obstacle's threat exceeds the threshold, then the detected obstacle is deemed threatening and is passed to the path-planning algorithm for consideration. The road bounds, required by the path-planning algorithm, are also determined from the set of detected obstacles by searching for small cone-sized obstacles spaced equidistance from each other at a predetermined spacing. Detection of the road boundaries is critical as this provides an estimation of the vehicle's yaw angle with respect to the roadway and constrains the path-planning algorithm so that no paths are planned that venture off of the roadway. Once two cones are identified, a line is virtually drawn through them to represent one side of the roadway while the known road width and estimated yaw angle is used to offset this line in order to determine the other side of the roadway. Relying solely on the LiDAR's point-cloud data to determine and differentiate between cone objects and hazardous obstacles simplifies the software. . While this is not an ideal approach, this is a compromise that avoids implementation of additional sensors, image processing, and/or sensor fusion, which may be added in the future. An example of the results given by the segmentation algorithm is shown in Figure 21.

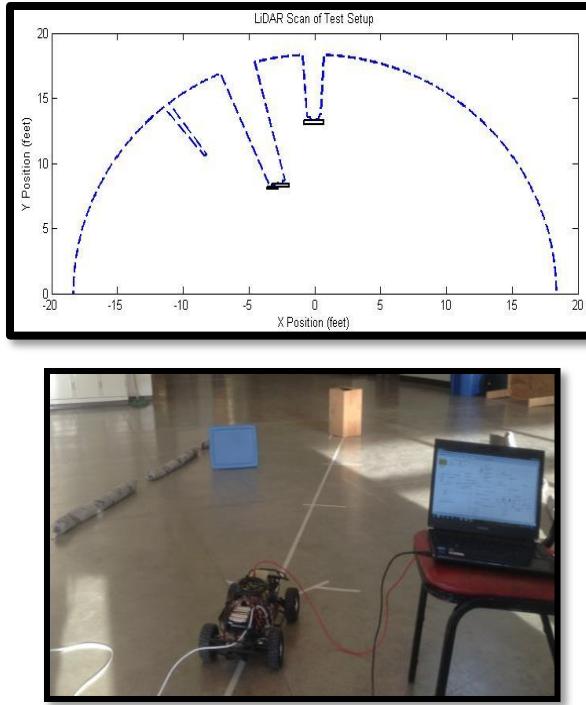


Figure 21. Lidar scan (blue) with parsed obstacles (black rectangles) and photo of scanned area

### 2.1.11 RRT and Path-planning

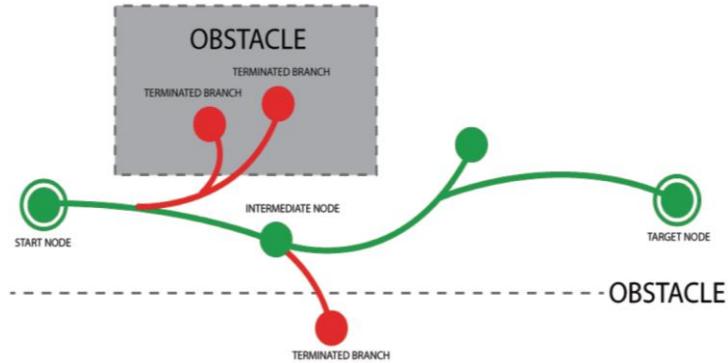


Figure 22. RRT nodes and branches mapping valid paths (green) and failed paths (red) in the driving plane

To generate a smooth path which the vehicle attempts to follow using the pure-pursuit control law, a path-planning algorithm must be implemented that takes into consideration the obstacles, the road bounds, and the velocity information obtained from

the sensory equipment. This path must take into consideration the non-holonomic constraints associated with vehicle dynamics; a car cannot rotate about its axis without also changing its position (Michielsen, 2013). The path-planning must provide temporary and reactive maneuvers in an attempt to avoid any collisions while navigating toward a target goal. Many path-planning algorithms exist in order to achieve this such as genetic algorithms, fuzzy logic, and artificial neural network algorithms. These methods attempt to find an optimal path and are generally very complex and pose a high computational overhead that make them unsuitable for real-time applications. The 2010 “Autonomous Crash Avoidance System” senior project adopted the Rapidly-exploring Random Tree (RRT) algorithm for this system (Miller, Ujiie, & Woods, 2011). The RRT method begins by generating random branches from a starting node. Each branch is then compared with the boundary conditions defined by the detected obstacles and road bounds in the driving plane. If the branch does not meet the boundary conditions, the branch is deemed unsafe and is terminated. If the branch does meet the boundary conditions, it becomes an intermediate node and new random branches are generated and compared with the boundary conditions until a viable path is found that reaches the target node. This process is shown in Figure 22 and the results of a safe planned path can be seen in Figure 23. This method makes no attempt to find an optimal path but is effective for real-time applications. As seen in Figure 23, the number and types of paths generated by the RRT algorithm can widely vary. Successive iterations of the RRT may plan paths in opposite directions or right on top of each other due to the random nature of the algorithm. Multiple iterations may search the same area for a viable path and there is no guarantee that the algorithm will sweep the entire driving plan while searching for a valid

path. Despite the somewhat random nature of the path planned by the RRT, if properly constrained and smoothed viable paths are usually found within a sufficient number of iterations that don't require excessive maneuvering. In addition, the first safe path found will be taken. Generally, more constraints passed to the path-planning algorithm can help bias the algorithm to quickly find safe paths through anticipated corridors. For instance, if two obstacles are placed at either side of the roadway the RRT will usually produce safe paths down the center of the roadway in between the two obstacles. However, there is no guarantee that a safe path won't be planned around the obstacle that nears the road boundary if the vehicle's size does not prohibit this. This is an example of how a path-planning algorithm could be optimized to find a safe path that complies with the minimum intervention principle by straying the minimum distance from the center of the roadway. On the other hand, by over-constraining the RRT algorithm no valid safe path will be found as only so many iterations may be performed before the deadline for actuation passes. Once a safe path has been planned to the goal, the vehicle attempts to follow it by using the pure-pursuit control law to calculate the required steer angles.

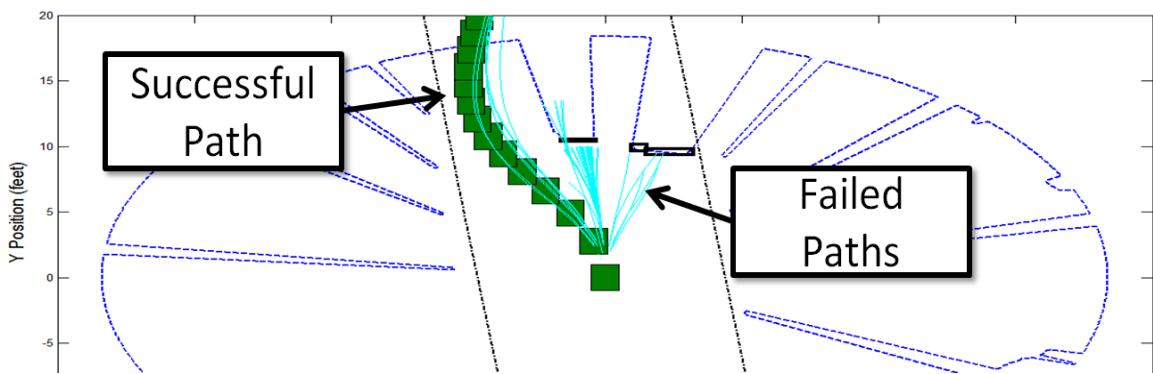


Figure 23. Sample of the results of the RRT algorithm

## 2.2 Fundamentals of Real-Time Embedded Systems

Embedded systems typically execute a program in a tightly constrained loop in order to provide reactive control to changes in the environment. An embedded system must continually react to changes in the system's environment and perform all calculations in time without delay, especially in control applications. A common misconception is that real-time means fast, which isn't necessarily the case; real-time does not mean high throughput but rather high responsiveness. Real-time systems must have a high-level of determinism and meet operational deadlines by guaranteeing a system response from an event within strict time constraints. A real-time system typically is one that perceives environmental data, processes the data, and actuates a controlled response sufficiently quickly. However, in the context of simulation, real-time refers to the simulation's clock running as fast as a real clock. Real-time systems may be classified into three groups depending on the consequences of missing a deadline:

- Hard real-time: Total system failure results from missing a deadline
- Firm real-time: Missing a deadline infrequently is tolerable, but may degrade performance
- Soft real-time: The usefulness of a system response is degraded after missing a deadline

A single-board computer running embedded Linux can serve as an ideal platform on which to develop an embedded controller for semi-autonomous collision avoidance. Embedded GNU/Linux is widely used as the operating system and kernel, respectively, that is deployed to thousands and thousands of embedded devices (Free Electrons). The operating system is a layer of software that provides low-level services to the application

layer, which is a set of one or more programs executing on the CPU and consuming and producing input and output data. Described in (Vahid & Givargis, 1999), the operating system is responsible for deciding what program to run next on the CPU and for how long. The CPU is shared by a number of executing programs. The Linux scheduler bounces back-and-forth between processes, scheduling each process or task according to their needs when it determines to do so. This is called process/task scheduling and is determined by the operating system's scheduling policy. Meeting real-time constraints for all processes required for semi-autonomous collision avoidance is a demanding prospect for a personal computer, not to mention for an embedded single-board computer. While embedded Linux does not realize true hard real-time operation, or hard time-based determinism, soft real-time can be achieved and is suitable for an embedded real-time control system. This refers to the notion that the kernel attempts to schedule processes within timing deadlines but makes no guarantees. Linux supports several scheduling policies: SCHED\_OTHER, SCHED\_BATCH, SCHED\_IDLE, SCHED\_FIFO, and SCHED\_RR. There are two real-time scheduling systems for the Linux scheduler: the first-come first-served (SCHED\_FIFO) and the round-robin (SCHED\_RR). By changing the default Linux scheduler to use the round-robin (SCHED\_RR) real-time scheduler policy, processes or tasks are given time-slices based on priority and the operating system schedules each one for execution. The round-robin scheduler gives real-time processes at a given priority level turns running for their issued time quantum. Tasks with a higher priority may preempt tasks with lower priorities. Preemptive scheduling of tasks may make the exact sequence of task execution difficult to determine and is subject to change during runtime. There is a separate run queue for all the processes of each priority level

running on each processor. In other words, each processor has 140 run queues corresponding to each priority level; priorities 0-99 are for real-time processes and priorities 100-139 are for normal processes. The basic scheduling algorithm finds the highest-priority run queue with a runnable process, gets the first process within that queue, and calculates the process's quantum size, runs the process, moves the process to the expired list when its time is up, and then repeats. The Linux scheduler attempts to be as efficient as possible when scheduling processes in the processor's run queues. Recent updates in the Linux kernel have made Linux much more effective for real-time applications.

This section introduced the principles of real-time operation and three classifications of real-time systems. The Linux scheduler and the use of embedded GNU/Linux running on a single-board computer as a soft real-time embedded controller is presented. In a process or task based finite-state machine software architecture multiple tasks, each operating a finite-state machine, compete for time slices of the processor depending on their assigned priority to operate between finite states. Further description of finite-state machines and their implementation follows in the remaining sections.

### 2.2.1 Finite-State Machines, Multithreading, and Thread Safety

Finite-state machines (FSMs) are a method of developing mathematical system models where the output depends on the entire history of inputs and previous states. A FSM's behavior is composed of a finite number of states which may produce actions. A FSM can only be in a single state at once and will transition to other predetermined states when a triggering condition or input event is observed; the transitions of a FSM are often presented graphically in a state-transition diagram that shows all possible states and the

conditions that would cause transition. FSMs are widely used to develop and design computer programs, especially for embedded real-time applications. A FSM based software design and event-driven programming techniques are efficient in handling asynchronous events (Dash, Dasgupta, Chepada, & Halder, 2011). An example finite state machine is shown in Figure 24.

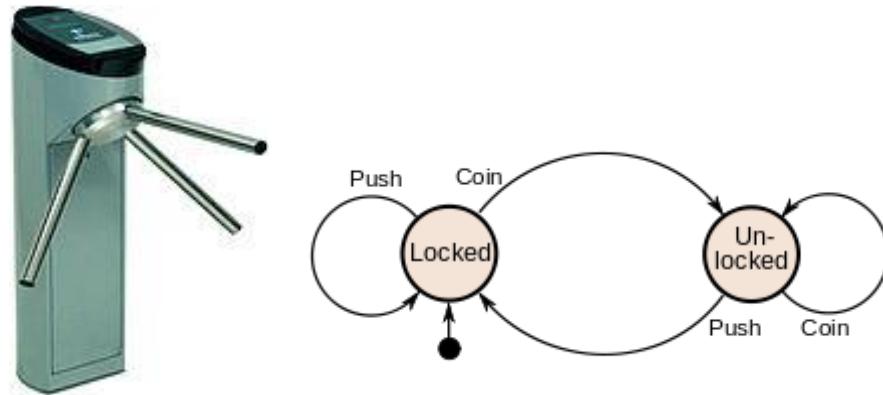


Figure 24. Example State Transition Diagram for a Turnstile Finite-State Machine

This turnstile example shows the basic principles of how a finite-state machine operates. The turnstile can be in either of two states: locked or unlocked. Two inputs affect the turnstile's state: pushing the arm and inserting a coin into the coin slot. The turnstile FSM starts in the locked state. When a coin is deposited, the turnstile transitions into the unlocked state and only a customer pushing through the arms will return the turnstile to the locked state. In the locked state, the turnstile will only shift to the unlocked state if a coin is inserted. Pushing the turnstile while in the locked state and putting in more coins while in the unlocked state has no effect on the turnstile's state.

This project uses a digital computer to implement a controller that runs multiple FSMs, concurrently, in separate threads. Digital computer control systems have become a prominent method of discrete control implementation due to the low-cost of microprocessors, the flexibility of the controller configuration and software, the

scalability, the adaptability in which a program can adapt or change with time, and digital controllers are much less prone to environmental factors such as temperature and humidity. FSMs are among the most used building blocks in embedded system design for creating firmware. FSM event driven programming is an appropriate fit for a reactive, event driven, real-time controller as most embedded systems respond to internal or external events such as an external interrupts, a change of signal level at I/O pins, a serial data packet, etc. The main advantage of using state machines in embedded system software is the flexibility to add, delete, or change the flow of the program without impacting the overall system code structure. This project ports a Simulink controller into a task based software architecture in order to run the collision avoidance firmware on a single-board computer (SBC). The duties of each task are encoded as Finite State Machines (FSMs) and the resulting embedded software architecture becomes a system of continuously executed FSMs. The SBC operates the controller that collects data, determines the threat, plans safe paths, and actuates hardware in real-time by concurrently running multiple event-driven, task-based FSMs across multiple threads. By running each task's FSM in a separate process or thread, known as multithreading, the Linux scheduler handles scheduling each task for execution and performing context switches between operating each task's FSM according to the real-time round-robin scheduler, described above.

For real-time simulation or real-time embedded systems with many tasks to be performed concurrently, a divide and conquer approach can be taken. In the context of a real-time embedded system, a task may be generically defined as a separate process that usually contains an infinite loop and runs based on its priority. A task may or may not

access hardware resources or shared data with other tasks. In an embedded real-time system with a multitasking or multithreading software architecture, each task is run in a separate thread where a scheduler addresses them concurrently by switching between them according to the scheduling policy; this is referred to as context-switching. A thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system. This means that many procedures running independently from the main program can be scheduled by the operating system to run concurrently and independently. This can be implemented in C++ on embedded GNU/Linux systems with the use of POSIX threads. Programs that use multiple threads are referred to as parallel programs. Due to the potential to damage the shared data between tasks, as many tasks may read and write to it, some synchronization strategy must be carefully implemented and tested. Since the scheduler switches contexts in order to run each task in parallel, each task must be written so that code reentry is safe and execution may safely resume from the last instruction executed in the task. As many tasks share the same resources or data, there is a potential for multiple tasks to attempt to write to or read from global variables simultaneously. This prompts the need to incorporate some form of thread safety such as through the use of semaphores or mutual exclusions (mutexes). These subroutines deal with synchronization by providing read/write locks and barriers required to maintain thread safety.

Multithreading allows for a modular implementation and realization of real-time operation. Each task has a direct correspondence with a thread where the FSM for that task is implemented. The tasks share data using a global shared memory model shown in Figure 25. In addition, depending on the selected hardware more than one thread may run

simultaneously depending on the number of available CPU cores. Tasks are very modular as are the threads that implement them; they can be added or subtracted independently of other tasks and their priorities and frequencies may be changed independently. Using threads to implement a task diagram with accompanying FSMs is a common method used to implement real-time systems on embedded Linux (Free Electrons) (Palopoli) (Vahid & Givargis, 1999). The controller's firmware used in this project is based on this approach. The design and implementation of the multithreaded FSM firmware is detailed in chapter 3.

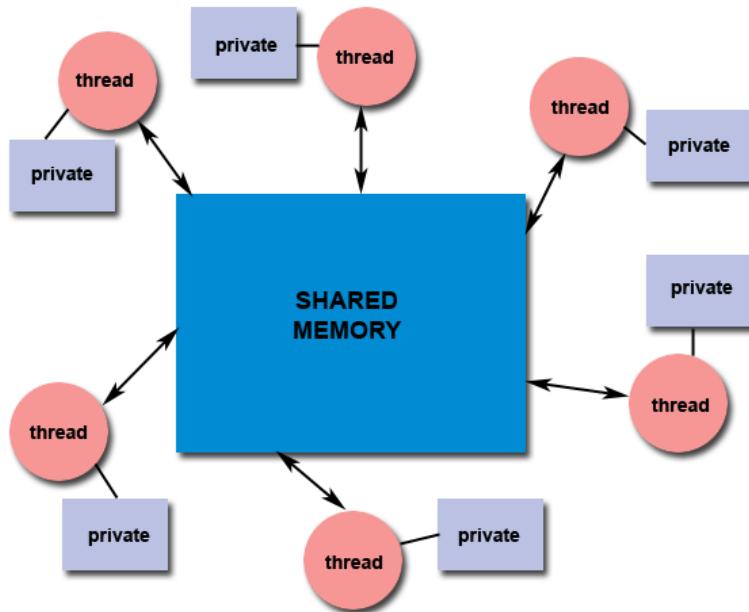


Figure 25. Shared memory model

## 2.3 Fundamentals of Simulation and GUI Development

### 2.3.1 Java Swing and Java2D API

When developing a graphical simulation, game, or application java has many application programming interfaces (APIs) that include sets of routines, protocols, and tools that ease development. One such API used exclusively to develop the user-control-

station GUI and the hardware-in-the-loop-simulator was Swing. Swing is a GUI toolkit developed for Java to provide graphical user interface (GUI) components for Java programs. This framework, along with the Java 2D API, provides the developer with containers, graphic primitives such as lines, circles, and rectangles, and other familiar GUI components such as check boxes, labels, and buttons.

### 2.3.2 Active vs. Passive Rendering

When a developer takes complete control over rendering by rendering graphics in a tightly controlled loop, this is referred to as active rendering. Not all applications implement or need active rendering and the question of when to paint the graphics may be more appropriately handled by the operating system. However, this can introduce a lot of system overhead and may deliver the rendering events at inappropriate or unpredictable times. For real-time simulations and game graphics the developer generally wants to decide when and how often to render the graphics and to draw directly to the screen. Active rendering not only provides complete control over rendering and the rendering engine, but has the advantage of being able to produce more frames per second which allows for smoother animations and sprite movements. It is the fastest way to render your program graphics and allows for consistent frames per second that is required for smooth animation. Typically, games and simulations aim to hit 60 noticeable frames per second, which is the max refresh rate of most monitors and any faster is usually said to be unnoticeable. The hardware-in-the-loop simulation developed for this project uses active rendering to maintain complete control over when and how rendering occurs.

### 2.3.3 Image Blitting

A bit blit is used in computer graphics to combine several bitmaps using a raster operator. This technique is used within the Java 2D graphics API and allows several sprites to be overlaid on a background and moved around according to the physics engine. In computer graphics, a sprite is a two-dimensional image or animation that is integrated into a larger scene. An example is shown below in Figure 26 in which a character sprite is added to the background scene and may be moved around using the physics engine written for character movement. In the simulation developed as part of this project, car sprites move around a roadway scene according to the vehicle's physics engine that is composed of the bicycle model.



Figure 26. Example of bit blitting used in computer graphics

## 2.4 Pin Multiplexing and Device Tree Overlays

System on a chip (SoC) integrated circuits (ICs), such as those used in the BeagleBone Black SBC, include many hardware blocks that interface to peripherals through pins (Free Electrons, 2015). By incorporating several hardware blocks on the same IC, greater functionality is achieved with a reduced number of required IC's that results in a more compact and low-power device. They may include many peripheral devices such as timers, analog-to-digital converters, serial communication devices, etc. By providing access to the physical IC pins, programs may use these peripheral devices

to monitor sensors, set actuators, and transfer data with other devices. Pins provide the interface for the processor to communicate with its peripherals and memory; they are the physical conducting device on the periphery of a processor through which a signal is input to or output from a processor (Vahid & Givargis, 1999). Due to the limited physical size of these small chips there exist only a limited number of pins. The pins are multiplexed to operate in different modes that expose their functionality for different hardware blocks and are used to accommodate the largest number of peripheral functions in the smallest possible package. For this reason, not all of the internal hardware features of the SoC may be used simultaneously as there may be conflicting pin assignments. Multiplexing acts much like a railroad switch that allows only one of multiple track inputs to connect to a single output track. A diagram below shows an example of how pin muxing operates. Multiplexing is used in this project to configure the processor's peripherals to expose the hardware that is used to control the motor and steer servo and to facilitate serial communication. A brief explanation of motor control and serial communication follows in the next section.

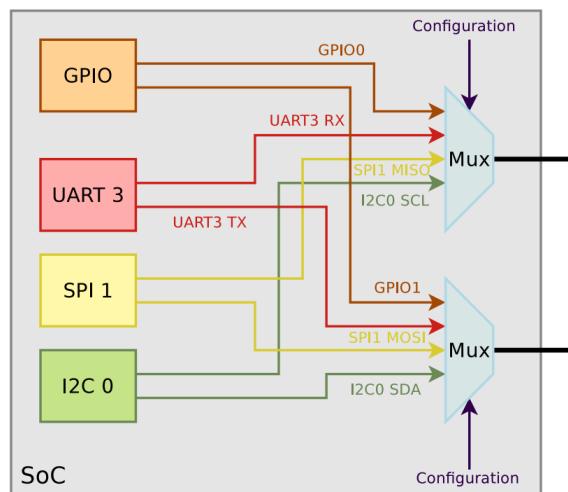


Figure 27. Example Pin Muxing or Multiplexing Diagram

## 2.5 PWM and Servo Control

As described by Vahid and Givargis, a pulse-width modulator generates an output signal that repeatedly switches between high and low. The duration of the high and low portion of the output signal is called the duty cycle and is the percentage of time that the signal is high compared to the signal's period. Many SBC's feature dedicated PWM digital peripherals. A common way to use PWM to control hardware components is to encode the control commands within the signal. This is how radio-controlled (RC) vehicle's motors and steer servo's are controlled. For example, a 10% duty cycle may correspond to a left turn command while a 20% duty cycle corresponds to a right turn. Off the shelf RC vehicles use radio control receiver & transmitter pairs to relay control signals to the motor's electronic speed controller (ESC) and the steering servo from the driver. The radio control receiver demodulates these signals and delivers an encoded PWM control signal that dictates how the motor and servo respond. In order to use an RC vehicle for a semi-autonomous collision avoidance system, the onboard controller must translate the driver or computer control steer and throttle commands into PWM signals that the motors can understand.

## 2.6 UART Serial Communication

Described by Vahid and Givargis, a Universal Asynchronous Receiver/Transmitter (UART) receives serial data and stores it as parallel data and takes parallel data and transmits it as serial data. This form of serial communication can communicate bytes of data between devices separated by long distances while requiring only a few I/O pins; two control signals are required, one for transmitting data (Tx) and one for receiving data (Rx), in addition to a ground reference. In order to use UART the

baud rate that indicates the frequency that the signal changes must be configured for both transmission and reception. In UART, bytes of data are transmitted as a stream of 1's and 0's (8 bits per byte) at a predetermined baud rate between the host and target. The transmitted bits are received and stored in the receive data register until all the bytes in the transmit register are processed. A flag bit alerts the main program occurs when the receive register is full. The data that is transmitted must be broken into discrete and recognizable chunks that can be reconstructed on the receiver side to obtain the original data without corruption. Single-board computers often are equipped with UART serial communication hardware that can communicate with other processors and peripherals. However, UART may be implemented completely in software. In this project, the UART peripheral on the SBC is used to facilitate communication between the user control station and the vehicle. The next chapter details the development of the embedded vehicle controller and its software.

## **Chapter 3     Developing the Semi-Autonomous Collision Avoidance System and an Embedded Vehicle Controller**

In order for a vehicle to semi-autonomously avoid collisions it must be able to sense the surrounding environment, identify obstacles, assess threats, and react to these threats by controlling the steering and/or throttle signals in the case of an imminent collision. These are very specific and consistent tasks that must be performed while adhering to real-time operating deadlines and are ideally suited to being performed onboard an embedded system. So what is an embedded system? There is no readily available definition for an embedded system. As described in “Embedded System Design: A Unified Hardware/Software Approach,” an embedded system is essentially any computing system that is not a desktop, laptop, or mainframe computer. Embedded systems are very commonly seen today in products ranging from routers and ATMs to MP3 players and traffic lights that perform dedicated computing functions within larger mechanical or electrical systems, often with real-time operating constraints. Unlike personal computers that are designed to be flexible and perform a variety of tasks, embedded systems perform dedicated tasks that can be optimized to reduce cost and power consumption and to increase performance and reliability. As described by Vahid and Givargis, any embedded system’s functionality consists of three aspects: processing, storage (memory), and communication. Embedded systems typically employ a microcontroller or a microprocessor to implement processing and to control devices. A general-purpose processor, sometimes called a CPU, is a programmable digital system used to implement the system’s functionality. Single-board computers (SBCs) are complete computers built on a single circuit board with processor(s), memory,

input/output (I/O), and other features required for a functional computer and are great candidates for use as embedded controllers. They vary in size, but are typically smaller, lighter, more reliable, and more energy efficient than their multi-board computer counterparts. However, different economies of scale mean motherboards are manufactured by the millions while SBCs fill a specialized market resulting in much lower manufactured volumes and higher costs.

In the previous implementation of this project, the 1/10<sup>th</sup> scale Traxxas RC car model utilized a personal computer with long wired cables to run a real-time Simulink controller. The Simulink model facilitated the calculations and data processing required to run the autonomous collision avoidance algorithms and to communicate with the user control station. The vehicle interfaced to the PC through a long USB cable to obtain the LiDAR data and a long Ethernet cable to pass the data from the car for processing and back to the vehicle to actuate commands to control the vehicle appropriately. This scheme was inherently inefficient, computationally, and had many opportunities for improvement by relocating the CPU onboard the vehicle. The microprocessor embedded system aimed at replacing the large and over-scale personal computer with a more to-scale SBC that is designed and better suited to interface with low-level peripherals and hardware. This helped to reduce transmission times, but gave rise to other hurdles such as wireless communication with the user control station, limited processing power, RAM, and onboard storage space. As modern vehicles use many electronic control units (ECUs) for their operation, the use of an SBC as the vehicle's ECU better aligns with a production implementation. In addition, this eliminates the wired tether, which offers a longer range and better mobility. This enables for more complex driving scenarios to be

investigated while providing the driver with a more realistic driving experience. The ultimate goal of autonomous/semi-autonomous collision avoidance systems is to greatly improve vehicle safety and to reduce vehicle related injuries by mitigating vehicle collisions. Research aims to develop autonomous vehicle controllers that are capable of this and then to implement them widely into the automobile fleet; in order to achieve this not only does a physical prototype need to be developed, but a dedicated and scaled CPU must also be developed alongside using common low-level hardware which exhibits the minimum amount of CPU, power, and cost required to run autonomous collision avoidance software. The selection of a single board computer along with the porting and implementation of the embedded system firmware follows.

### 3.1 Single-Board Computer Selection and Evaluation

There are many commercially available SBC development boards that are suitable for prototyping embedded systems. These boards are being continually improved to higher levels of capability and speed. As IC capacities grow so do the complexities of the embedded systems that incorporate them. Fortunately, many SBCs are host to large open-source communities, support forums, and code/project repositories with useful algorithms and ideas for a variety of applications such as image processing, signal conditioning, and filtering. In order to evaluate the copious selection of available development boards, an evaluation/decision matrix was created that ranks each considered SBC against each other based on specific comparison metrics relating to the project. This evaluation matrix can be found in Appendix F. From the board comparison and decision matrix, it is evident that none of the SBCs stands out among the rest; there is no wrong choice except for those boards that do not meet the minimum requirements and

specifications. All the SBCs are relatively inexpensive so deciding the tradeoff between cost and performance essentially held no significance. Many of the development boards are well suited and surpass the minimum needs of the project's embedded system. More powerful SBCs will continue to be developed while old ones are phased out as production ceases. One must be selected with future revisions/updates/versions in mind as well as continued community support in an effort to minimize the problems associated with software/hardware updates.

The BeagleBone Black was selected for this project as seen in Figure 28. This single-board computer is not only host to a large community support network with innumerable code libraries and references, but also places at the top in benchmark performance tests (Dhrystone test and Whetstone test). The BeagleBone Black is a highly extensive and flexible single-board computer with the capability to add many features and interfaces by incorporating custom circuit boards or 'capes' that access the processor's pins via the expansion headers. The BeagleBone Black features an AM335x ARM processor that runs at 1GHZ. Power may be provided to the board via USB, via a DC barrel jack, or via the 5V pin on the expansion header. There are certain limitations as to how many capes can coexist as the processor's pins are multiplexed to set the operating mode. The multiplexing of the processor's pins allows their use for a multitude of different peripherals, including GPIO, SPI, I2C, UART, CAN, LCD, PWM, etc. In addition to the main CPU, the BeagleBone features two completely independent programmable hard real-time units (PRUSS) and can share data with the main CPU. The operating system may be located on a bootable micro SD card or may be located on the

onboard eMMC memory. This allows for split second changes of operating systems and software for different applications; portability is a very useful prototyping feature.



	Feature
Processor	Sitara AM3359AZCZ100
Graphics Engine	1GHz, 2000 MIPS
SDRAM Memory	SGX530 3D, 20M Polygons/S
Onboard Flash	512MB DDR3L 606MHZ
PMIC	2GB, 8bit Embedded MMC
Debug Support	TPS65217C PMIC regulator and one additional LDO.
Power Source	Optional Onboard 20-pin CTI JTAG, Serial Header
PCB	miniUSB USB or DC Jack
Indicators	5VDC External Via Expansion Header
HS USB 2.0 Client Port	3.4" x 2.1"
HS USB 2.0 Host Port	6 layers
Serial Port	1-Power, 2-Ethernet, 4-User Controllable LEDs
Ethernet	Access to USB0, Client mode via miniUSB
SD/MMC Connector	Access to USB1, Type A Socket, 500mA LS/FS/HS
User Input	UART0 access via 6 pin 3.3V TTL Header. Header is populated
Video Out	10/100, RJ45
Audio	microSD , 3.3V
Expansion Connectors	Reset Button Boot Button Power Button
Weight	16b HDMI, 1280x1024 (MAX)
Power	1024x768,1280x720,1440x900 w/EDID Support
	Via HDMI Interface, Stereo
	Power 5V, 3.3V , VDD_ADC(1.8V)
	3.3V I/O on all signals
	McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7
	AIN( <b>1.8V MAX</b> ), 4 Timers, 3 Serial Ports, CAN0,
	EHRPWM(0,2), XDMA Interrupt, Power button, Expansion Board ID
	(Up to 4 can be stacked)
	1.4 oz (39.68 grams)
	Refer to Section 6.1.7

Figure 28. BeagleBone Black SBC and features

GNU/Linux is the most widely used operating system for embedded systems that minimizes cost and enables extensive modification. Many popular Linux operating systems have been ported to the ARM processor that sit atop the BeagleBone such as

Debian, Arch Linux, Angstrom, Ubuntu, etc. Most often, operating systems for embedded systems are stripped down versions of their personal computer counterparts containing little more than the Linux kernel and a few specialized processes in order to reduce required memory space and to allow greater energy efficiency. An operating system (OS) provides the basic set of programs and utilities that make a computer run. At the core of the OS is the kernel, which is the most fundamental program on the computer that manages all the hardware resources and provides a set of portable APIs that are hardware and architecture independent to allow user space programs access the hardware resources; the Linux kernel connects the system hardware to the application software. With appropriate scheduling, embedded Linux can realize soft real-time operation. Recent advances in the Linux kernel have improved Linux's real-time capabilities. The OS selected to run the vehicle controller was Debian GNU/Linux with version 3.8 of the Linux kernel. Debian is a completely free OS that is widely used by large and small organizations alike. Debian is also the OS that currently comes shipped on the BeagleBone Black.

This section detailed the selection of a SBC to replace the PC that ran a real-time Simulink controller. An evaluation table was constructed with various comparison metrics that lead to the selection of the BeagleBone Black SBC to run the semi-autonomous collision avoidance controller based on the project requirements. The remaining sections detail the design and porting of the existing PC's software to embedded Linux as well as the integration of the system's software and hardware.

### 3.2 Porting of existing code from Matlab/Simulink to C++

The Matlab/Simulink model, pictured in Figure 29, is a high-level block diagram that previously ran the collision avoidance software. This software is composed of blocks and code that collects data from the LiDAR sensor, segments and parses the data, determines a threat metric, plans a safe path using the RRT algorithm, and actuates the appropriate controller or driver response depending on the threat level. Each block within the diagram contains underlying code that is written in a variety of languages, including m-code, Java, and C/C++ and uses various built-in MATLAB functions such as those for random number generation. In (1), data is collected from the LiDAR sensor and is passed into the segmentation and threat determination (2) where the obstacles and road boundaries are parsed. Encoder data is retrieved from the serial bus (3) and used to determine the vehicle's speed. If no threat is detected in (2), then the driver's steer and throttle commands (6) are passed to the vehicle. If a threat is detected then a safe path around the hazard(s) is determined by the RRT algorithm (5). The toggle between driver and computer control is managed by a switch block (7). The steer and throttle commands are then sent back across the serial bus in (8). This block diagram provided the roadmap to translate the control loop into (soft) real-time embedded Linux software written in C++ that preserves the control loop. This was accomplished by devising a multitasking/multithreading software architecture that concurrently runs multiple tasks, encoded as FSMs. To implement this multitasking software architecture, multiple POSIX threads (Pthreads) were created with different frequencies and priorities to run all the tasks associated with collision avoidance. The blocks from the diagram in Figure 29 were converted into task-based FSMs and certain blocks were combined in relation to what

task the FSM is responsible for performing. For instance, gathering the LiDAR data (1), segmenting the point-cloud, determining the threat (2), and planning a safe path (3) were all combined into an avoid collisions task. As a consequence of using a SBC and removing the previous custom microcontroller and personal computer, a new wireless communication link had to be devised and all the sensor drivers needed to be rewritten. In addition, the underlying code for each block in Figure 29 had to be ported to a language understood by the SBC so that it may be incorporated into the appropriate tasks.

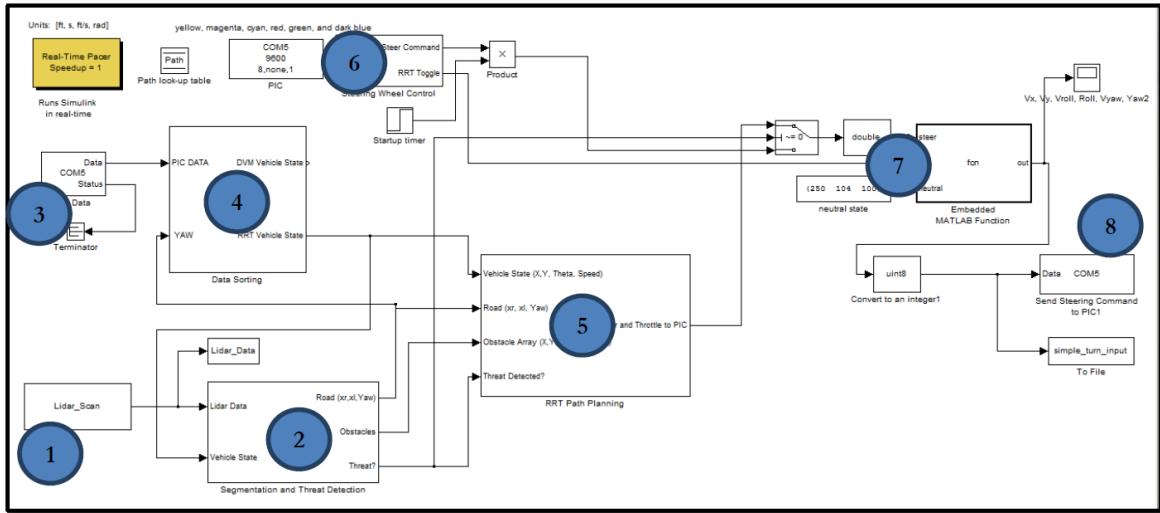


Figure 29. Matlab/Simulink semi-autonomous collision avoidance software model

The porting of the existing code was largely syntactical and time-consuming. This required familiarity with all utilized programming languages so that similar facilities could be used when porting the code from one language to another. All of the code was ported by hand to C++ to reduce convolution and bloated automatic conversions, ensure comprehension of ported code for future revisions, and to be as efficient as possible. An important aspect of the port consisted of deciding how to represent data in containers and finding sufficient datatype equivalents across programming languages. The ArrayLists used in the Java programming language had an affinity to vectors used in the C++

programming language. In Matlab/Simulink, it was found that matrices were most easily translated into vector of vectors which allows for easy manipulation and dynamic capabilities. Matrices with fixed sizes were translated to multi-dimensional arrays or single dimensional arrays by appropriate manipulation of the indices. One critical aspect of the port was considering that array indexing in Matlab begins at one while array indexing in C++ begins at zero. Another challenge associated with porting the RRT path planning algorithm was generating uniformly distributed random numbers. The algorithm which Matlab uses to generate random numbers is proprietary. The Mersenne Twister pseudorandom number generator was used in this project to generate random nodes within the RRT algorithm. It is a widely used random number generator that was designed to rectify flaws found in older random number generators and provides reliable, high-quality, fast random number generation. The Matlab and C++ software was stepped through line-by-line and compared to ensure that the same output was produced when provided with the same seeds, boundary conditions, and input data. A Matlab program was also created to aid in this process by graphically plotting the results of the embedded software to visualize the output. With the underlying Simulink controller code ported to C++, the software architecture was restructured. The restructuring and design of the embedded software architecture follows in the next section.

### 3.3        Embedded System Software Architecture

In order to utilize the SBC's processor, the embedded system designer must write a program. Embedded programs can incorporate the use of multiple threads executing concurrently to maximum use of the CPU, reduce CPU idle time to a minimum, and handle urgent requests immediately. This configuration yields a software architecture

where multiple tasks cooperate together in order to run the collision avoidance system.

The embedded system's processing behavior is dictated by the concurrently running tasks. The duties of each task are encoded as a FSM and the resulting embedded software architecture is a system of periodically executed FSMs. Embedded systems typically employ FSM models in a control-dominated system to constantly monitor control inputs and react by setting control outputs. In FSM models, introduced in chapter 2, the system behavior is described as a set of possible states and the system may only be in one state at a given time. All possible transitions from one state to another are described as well as any actions that occur while in a state or when transitioning between states. The behavior of each task may be described through the use of a sequential program as detailed by Vahid and Givargis. C++, a high-level programming language with sequential execution in which one statement executes at a time, was used to write each task's FSM. This provides a set of objects, rules for object composition, and execution semantics of the composed objects that is used to describe the embedded system's behaviors. Object oriented modeling and object oriented programming provides an elegant means for describing complex systems by breaking the system into smaller and simpler, well-defined objects. The C++ programming language is used to capture the model in concrete form by creating an object model for each task. An object model is also created for the LiDAR sensor and data processing algorithms. The general form of a task is shown below:

```
void * PeriodicTask(void * arg) {  
    <initialization>;  
    <start periodic timer, period = T>;
```

```

while (cond) {

    <task_body>

    <wait next activation>;

}

}

```

The embedded real-time software is composed of a series of finite state machines that operate together to realize semi-autonomous collision avoidance in the case of an imminent collision. Tasks are implemented in multiple threads where each finite state machine operates independently. A common way to share a processor among multiple processes is to rely on a multi-tasking operating system to schedule processes for processing, allocating storage, and interfacing to peripherals. The threads or tasks communicate using a shared memory model to allow the exchange of data. A thread is a lightweight sub-process within a process that has its own data, registers, stack, and scheduling priority and shares system resources with other threads. Thread creation may be completed quickly and switching between threads by an operating system does not incur heavy overhead. POSIX threads were developed to support concurrent processes in sequential programming languages such as C and C++. An OS that specifies constraints on the rate of the processes and guarantees that they will be met is referred to as a real-time operating system (RTOS). The primary concern for each task is that it completes within its deadline; the scheduling priority of each task is selected to accomplish this. Real-time embedded systems maintain deterministic operation where the distribution of computation time is known to be within some bounds. In regards to real-time scheduling for tasks with harmonic periods, all tasks finish before their next activation if:

$$\sum_i \frac{\text{Computation Time}_i}{\text{Period}_i} \leq 1 \quad (31)$$

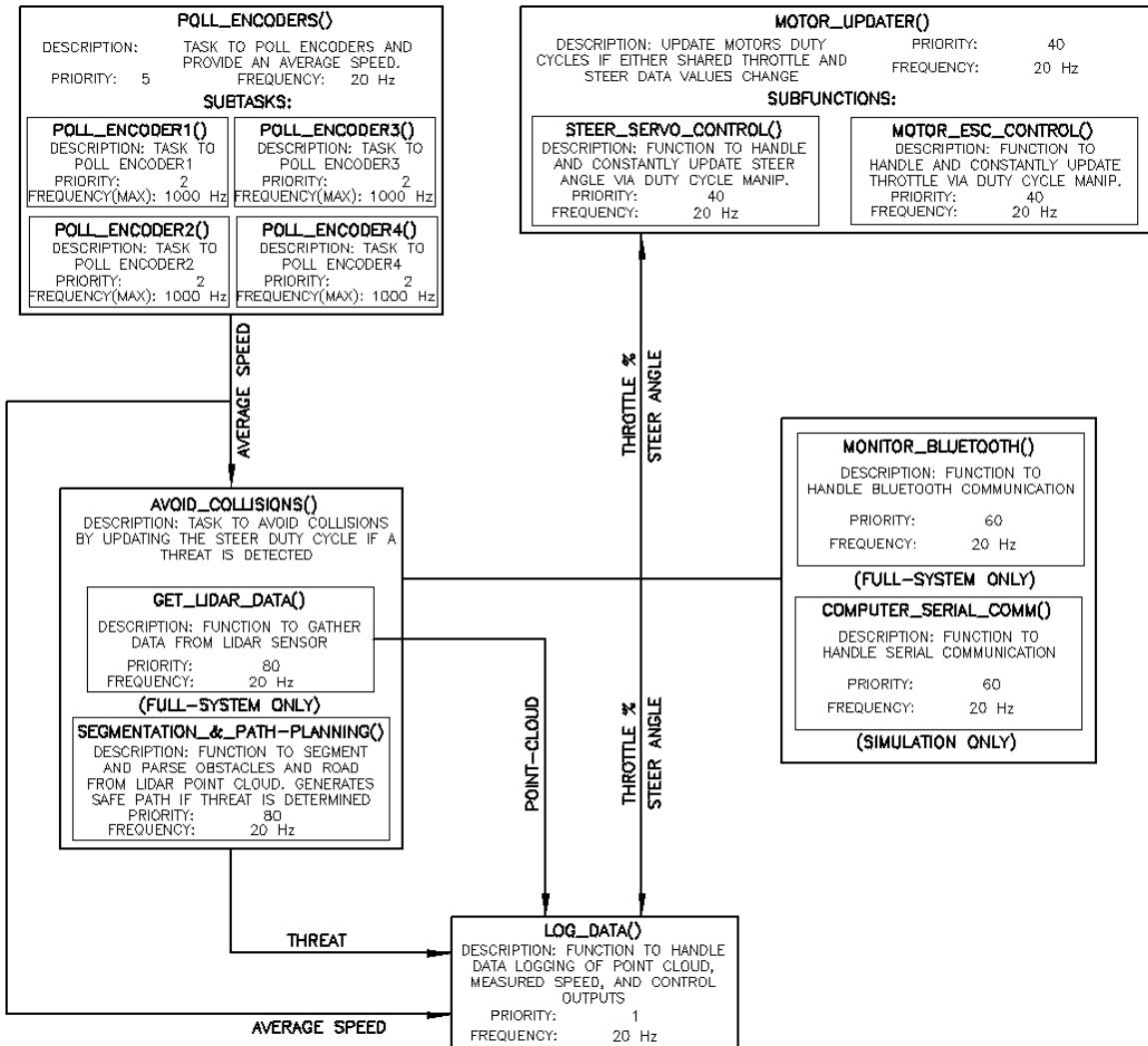


Figure 30. Real-time embedded software task diagram

The task diagram shown above in Figure 30 is the basis of the embedded controller's software architecture and shows all the tasks that the real-time system performs concurrently in order to achieve semi-autonomous collision avoidance behavior. The blocks from the Simulink model, in Figure 29, have been reconfigured, combined, and dispersed to form tasks with specific duties that are encoded and implemented as discrete FSMs in a multithreading/multitasking configuration. With this multithreading

scheme, each thread is synonymous with a task and a FSM. The Linux scheduler handles switching context between tasks to address their needs. The diagram shows the priority of each task as well as the frequency at which each task runs as well as the thread's priority that runs each task according to the Linux scheduler's real-time round-robin configuration described in chapter 2. Also shown is the flow of shared data between tasks. This shared data includes the sensor measurements, the estimated threat of a collision, the LiDAR's point-cloud, and the actuated control signals. Each task is implemented as a FSM that performs the duties described in the task's description. The *motor updater* task is responsible for updating the PWM signals that control the motor's ESC and the steering servo. The *poll encoders* task periodically averages all the wheel encoders counter variables to provide an estimation of the vehicle's average speed and then resets the counters. The *poll encoders* task creates a *poll encoder* sub task to monitor each wheel encoder using an interrupt based approach. With respect to the experimental ground based system, the *monitor Bluetooth* task is responsible for handling the serial communication and relaying the driver's commands to the vehicle from the user control station. For the simulation, the *monitor serial* communication task transmits the simulated data from the computer to the embedded controller for processing and actuates & transmits back the control signals to simulate the vehicle's response. The *avoid collisions* task is responsible for processing the point-cloud data to determine if there is threat of an imminent collision, parsing the obstacles and road boundaries, and planning a safe path around any detected hazards. The *log data* task logs the measured data and control signals. The vehicle platform logs the LiDAR data, the threat level, the RRT path-planning valid path flag, and the motor actuation signals which provide enough data to

reconstruct the scenario in simulation and to identify when the computer or driver's steer command was issued. The implementation of the task diagram occurs on the main thread within the C++ program. The main thread's sole duty is to initialize and start all of the tasks as separate threads with the appropriate priority and frequency. After this, the main thread terminates execution.

Once a task or thread has been designated a frequency and created, a FSM performs the task's duties by periodically transitioning between states. Generally, each task has strict timing constraints and must use high resolution timers or clock functions to regulate their frequency in order to run at the desired rate. The CLOCK\_REALTIME high-resolution clock provides the system with a real-time clock in nanoseconds that can be used to monitor and regulate each task/thread to comply with their desired frequency. This functionality is accessed within the C++ code that operates each FSM by including the 'time.h' header and the library is linked with the '-lrt' flag. This time regulation allows the varying amount of time it takes to run the code within the task's body to be taken into account. Regulation also allows a thread to be momentarily suspended once its duties have been completed within time and the deadline has not yet passed. Suspension of a thread allows the scheduler to be more efficient when switching context between tasks which haven't yet been scheduled. Time values are handled with the *timespec* structure and are a standard way to store time values for real-time processing (Lipari, 2009). An example of a time regulated while loop is shown below.

```
while (RUN) {  
    clock_gettime(CLOCK_REALTIME, &start);  
    <task body>
```

```

clock_gettime(CLOCK_REALTIME, &end);

diff = (requestEnd.tv_sec-requestStart.tv_sec)+(requestEnd.tv_nsec-
requestStart.tv_nsec)/1E9;

long diff_us = diff*1000000L;

long sleep_us = period_us - diff_us;

if(sleep_us > 0) {

    timespec_add_us(&end, sleep_us);

    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &end,
NULL);

}

clock_gettime(CLOCK_REALTIME,&end);

diff = (requestEnd.tv_sec-requestStart.tv_sec)+(requestEnd.tv_nsec-
requestStart.tv_nsec)/1E9;

cout<< "Task Elapsed Time: "<< diff << endl;

}

```

The *monitor Bluetooth* task handles the operation of the FSM that obtains the driver's steer and throttle commands from the user-control-station by monitoring the Bluetooth module's serial pins. This task begins by opening the serial port and setting the steer and throttle command to neutral (i.e. no throttle & zero steer angle). The task then periodically reads the incoming driver throttle and steer commands as two bytes of data. One byte signifies the throttle PWM value and one byte holds the steer servo PWM value. Upon a successful read the throttle and steer command are updated, however, the steer command is only updated if no threat is detected. Then the task returns to the wait

state until the task is once again due for execution. The threat variable is globally shared between tasks. If there is threat of an imminent collision, the user's steer command is simply neglected and only the throttle value is updated. If the read fails and times out then the control signals are set to neutral as this signifies a loss of communication with the user control station. Operating at 115200 bps, the lag incurred by transmission and actuation of the servo and ESC did not cause any blatant discontinuities in user control with a properly set task priority and frequency. To provide safety to this task or thread, mutexes were used while altering the shared steering and throttle data to prevent clobbering of the data between threads or creating race conditions.

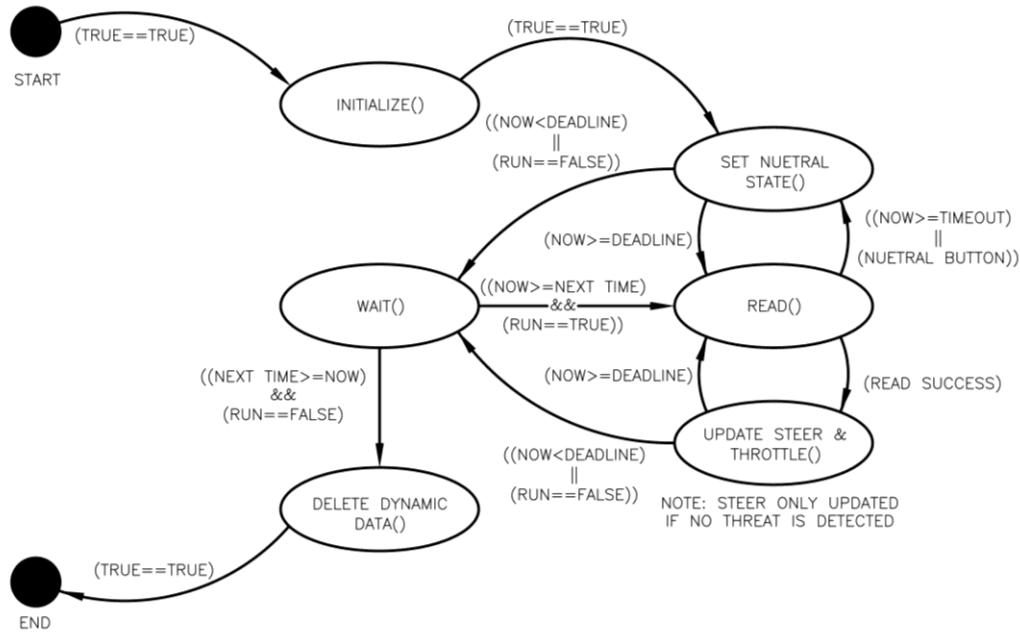


Figure 31. Monitor Bluetooth FSM state transition diagram

The *motor updater* task handles updating the control signals that actuate the ESC and steer servo responses. The task begins by initializing the mutexes, condition signals, and the PWM channels that produce the ESC and steer servo PWM control signals. Once initialized, the FSM waits until a change in the steer angle or throttle percentage is

required. This change is detected by using a condition signal that is shared globally with other tasks and allows them to signal that the PWM value of the steer servo or ESC has changed. Once signaled by another task/thread, the motors are updated by scaling the PWM value to determine the required duty cycle and setting the PWM channel accordingly, as shown below in Figure 42. Then the FSM returns to wait for another condition signal unless a shutdown request is issued. In the case of a shutdown request, the FSM deletes the dynamic pointer data and memory and closes the thread. This task does not consider whether the computer or the driver's control input is used to update the motors as this is determined in other tasks. This task merely waits for a condition signal that tells it to update the PWM control signals for the ESC and/or servo from the shared throttle and steer command variables. The frequency of this task is dependent on other tasks as it is only changes state and runs when the condition signal has been set by another task; this means the frequency of this task is determined purely by how often the control inputs are updated and changed by other tasks.

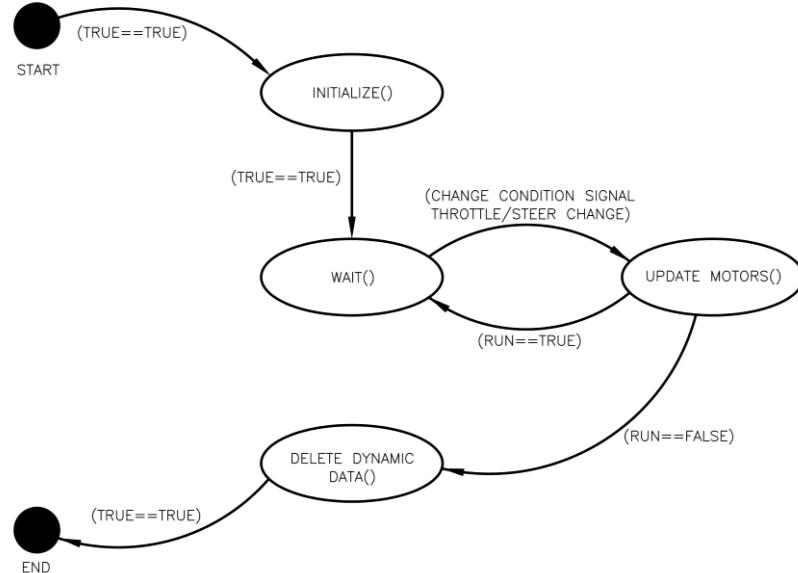


Figure 32. Motor updater FSM state transition diagram

The *poll encoders* task runs the FSM shown in Figure 33. This task begins by creating one *poll encoder* task or thread, as seen in Figure 34, to monitor each wheel encoder and to keep track of a counter; this creates four peer threads to handle monitoring each wheel encoder's counter variable. The *poll encoders* task periodically reads all the counter variables associated with each wheel's encoder, averages all the counter values together, and divides by the elapsed time to determine the average vehicle speed. After the vehicle's average speed is calculated, each wheel encoder counter is reset and the average speed variable, shared across all threads/tasks, is updated. Mutexes are used for each wheel encoder count variable and the average speed variable to provide safety from multiple threads attempting to read and write to the shared variables simultaneously.

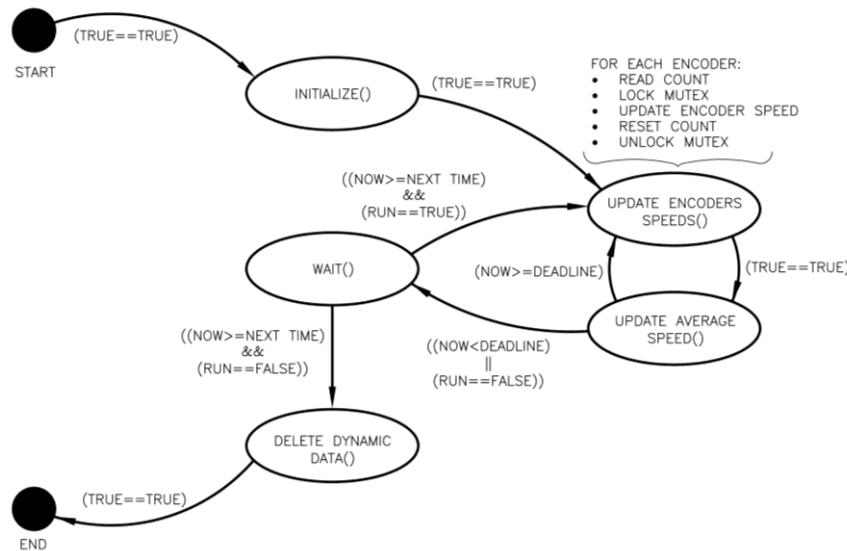


Figure 33. Poll encoders FSM state transition diagram

The *poll encoder* task is designed to monitor and keep track how many times a single encoder fires by incrementing a counter variable. This task incorporates the use of an interrupt to increment the counter as the wheel rolls. One poll encoder task, or thread, is assigned to each wheel encoder on the vehicle. The increment variable for each

encoder is used by the *poll encoders* task to calculate the average speed. By using an interrupt to run the FSM, shown in Figure 34, the frequency of the task completion can significantly vary. Once a wheel encoder interrupt is triggered, the counter will be incremented and the task will transition to a wait state. The amount of time to wait, if any, may be determined by considering how often each encoder should be polled so that no interrupts are missed. The required sampling rate establishes a deadline and how often the task may wait in between polls. According to the Nyquist-Shannon sampling theorem, the max frequency of this task can be calculated by using the maximum anticipated vehicle speed as follows:

$$\left( \frac{32 \text{ ticks}}{2\pi(0.1667 \text{ ft})} \right) \left( \frac{10 \text{ miles}}{\text{hour}} \right) \left( \frac{5280 \text{ ft}}{\text{mile}} \right) \left( \frac{1 \text{ hour}}{3600 \text{ s}} \right) = 448 \text{ Hz}$$

$$\text{Nyquist Rate} = 2(448 \text{ Hz}) = 896 \text{ Hz}$$

At this rate no encoder interrupt is missed and the data may be fully reconstructed while allowing the task to momentarily sleep. By allowing the task to wait or sleep once the task body has completed, the CPU is used more efficiently.

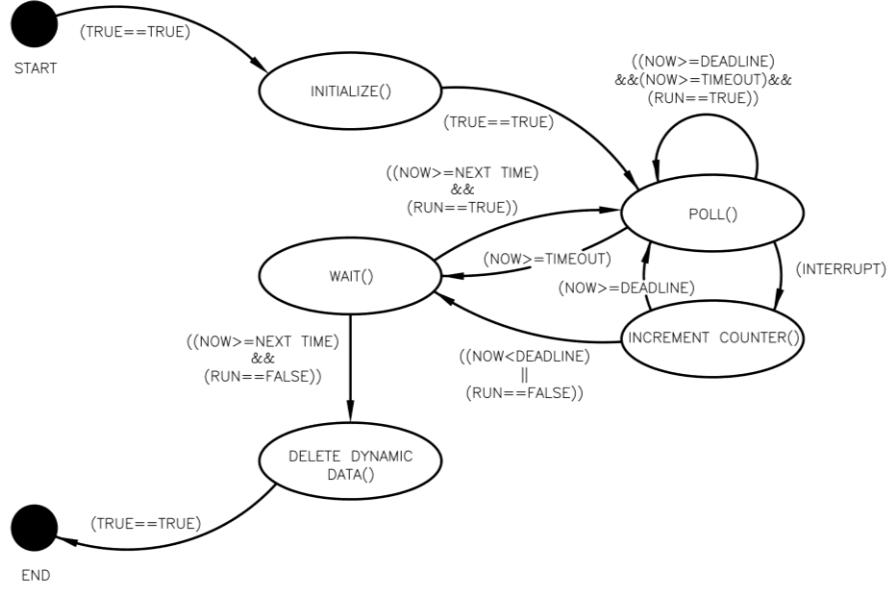


Figure 34. Poll encoder FSM state transition diagram

The *avoid collisions* task that implements the FSM shown in Figure 35 is the heart of the collision avoidance software. This task is responsible for obtaining and processing the LiDAR data in order to determine the threat of an imminent collision. In the case of an imminent collision, this task is also responsible for finding a safe path. The *avoid collisions* task has the highest priority because determining a safe path and the corresponding control outputs is of primary importance to an autonomous collision avoidance system. This task requires the most processing time compared to the other tasks and is given the highest priority to grant the largest CPU time slice with which to complete its duties. First, this task gets the point-cloud data from the LiDAR sensor. This data is then formatted, segmented, and parsed to identify hazardous obstacles and road cones within the point-cloud, as explained in chapter 2. Hazardous obstacles and road cones are then differentiated based on the detected size and spacing. The detected road cones are used to determine the road boundaries that the vehicle must navigate within and which obstacles may present the threat of a collision. If the road bounds can be identified

and the threat of an imminent collision exists, the obstacle and road boundary constraint data is fed to the RRT path planning algorithm in order to determine a safe path. The required control signals to follow the path are calculated and updated along with the update motor condition signal for actuation by the motor updater task. If a valid path is found the steering command is updated otherwise the driver will remain in control. The vehicle controller has a 50 millisecond control loop as was used in the previous revision of this system and was shown to be sufficient for autonomous control calculations by (Anderson, Peters, Pilutti, Tseng, & Iagnemma).

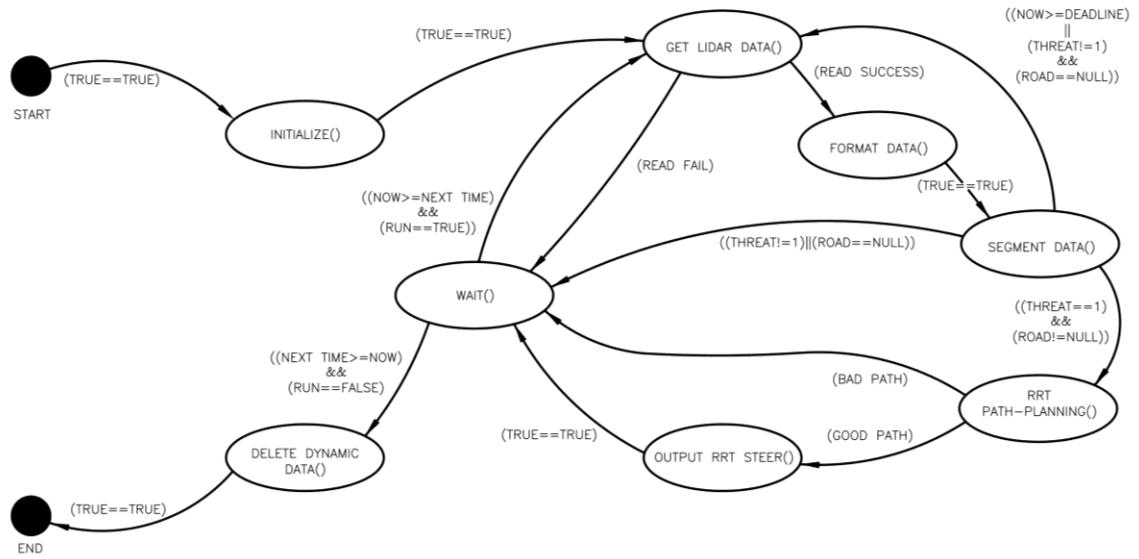


Figure 35. Avoid collisions FSM state transition diagram

This task incorporates the code ported to C++ from blocks (1), (2), and (3) in the Simulink model. A debugging solution was created to check that the output of the C++ code matched that of the Matlab/Simulink implementation for the segmentation and path-planning algorithms by passing in the same point-cloud data. Once each algorithm was ported to C++, a multitasking real-time software architecture was devised and each algorithm was incorporated into a task. Similar semi-autonomous collision avoidance

behavior was realized by porting to a real-time embedded system that runs a series of concurrent tasks that operate FSMs and communicate via a shared memory model. The modularity achieved with a multitasking architecture allows for both closed and open loop controller operation. Different modes of operation can be achieved by choosing which tasks run. The open and closed loop controllers operate by beginning the appropriate tasks and only the closed-loop controller starts the avoid collisions task. In order to run the software devised in this section the appropriate hardware must be selected and configured to interface with the sensors and peripherals used and the software must be compatible for the selected processor. The next section details the hardware implementation and electrical design in order to integrate the software for system operation.

### 3.4        Embedded System Hardware

The proposed system hardware architecture can be seen in Figure 36. This diagram shows all of the hardware both on and off the vehicle as well as the data flow between each hardware component. A PC is used to run the user-control-station but performs no data processing and merely relays the driver's steer and throttle commands to the vehicle, wirelessly, through a Bluetooth interface as small data packets.

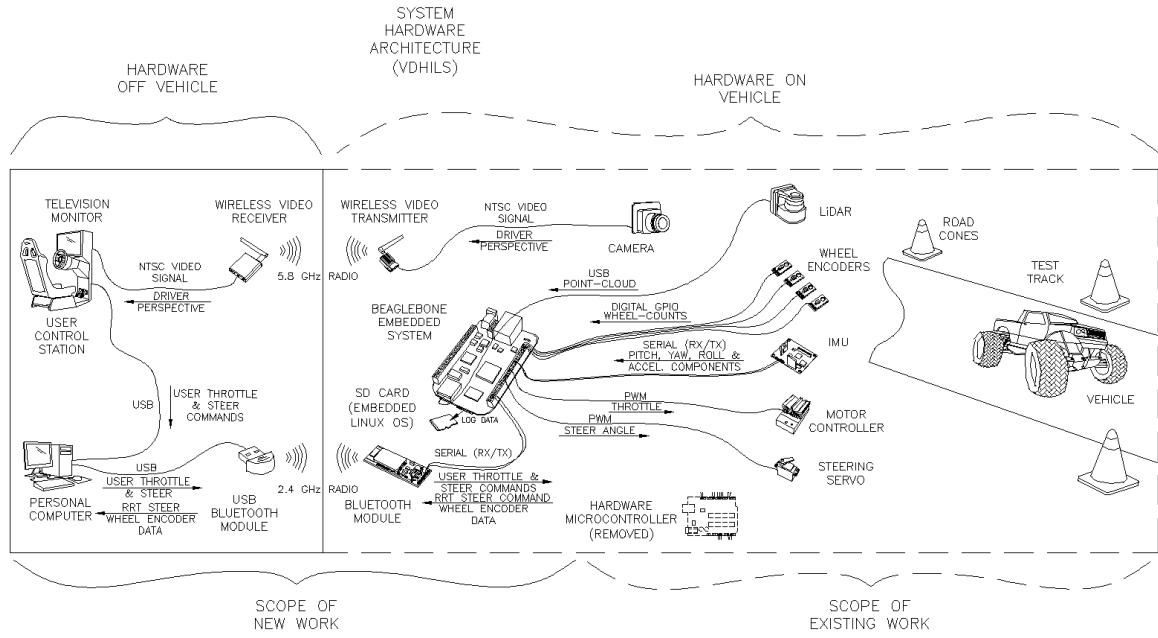


Figure 36. System hardware architecture

A custom printed circuit board (PCB) ‘cape’ was prototyped and fabricated to facilitate robust and easy connections between the hardware peripherals and the BeagleBone. In addition, a user interface (UI) was added to allow seamless switching between operating modes of the embedded software by pressing buttons on the cape. The UI features four LEDs for debugging, indication of operating mode & threat level, and two buttons for mode selection. Male headers are provided for connection to peripheral hardware and are described with a detailed silkscreen. Provisions were made to allow room for future development of system or the addition of sensor hardware by breaking out every connection of the P8 and P9 header of the BeagleBone with extended female headers. Reverse voltage protection was also added for both the 3.3VDC and 5VDC rails by utilizing P-channel MOSFETS to prevent the BeagleBone or any attached hardware from being subjected to damaging voltage levels. Solder bridges were applied to allow future development of the programmable real-time unit subsystem (PRUSS) to handle the

serial communication, PWM, and/or image captures. In the future, most of the sensor hardware could all be mounted directly onto the cape, including the IMU sensor, the Bluetooth module, and an FTDI integrated circuit (IC); however, this was not done in this revision in order to increase modularity of components. The BeagleBone was fitted with the cape and installed on the RC vehicle. The RC vehicle platform is shown in Figure 38.

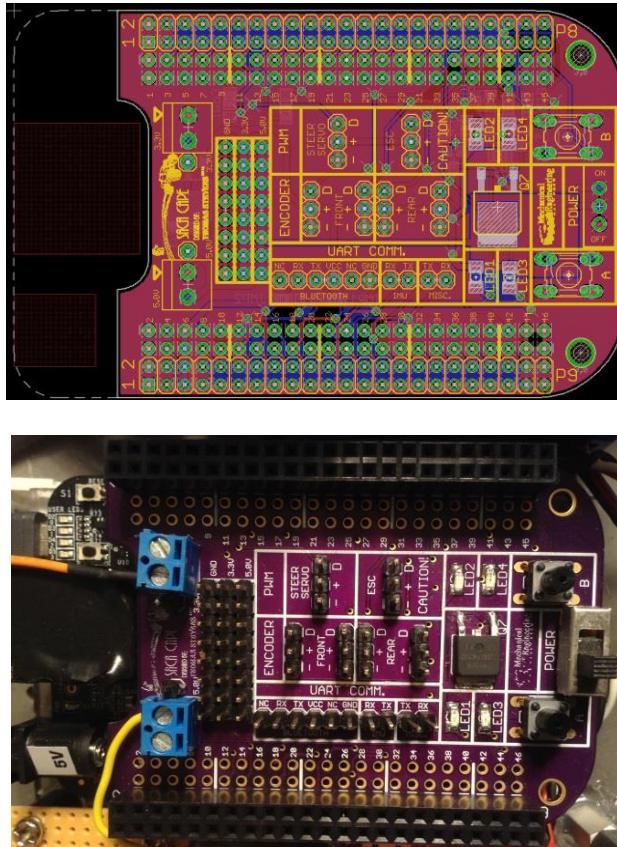


Figure 37. BeagleBone cape board design and populated board



Figure 38. 1/10th Semi-Autonomous Collision Avoidance Platform

The bill of materials for this system's hardware is shown in Table 1. Note that these costs do not include the user-control-station's hardware. Even with the inclusion of these costs, the total system cost is orders of magnitude less than developing a full-size experimental vehicle system. In fact, the total cost is an order of magnitude less than a full-size vehicle's sensor alone, such as the Velodyne LiDAR sensor used on the Google car. This is further justification for the scaled vehicle approach to developing a semi-autonomous collision avoidance platform in addition to the easier setup/maintenance. Most importantly, a scaled vehicle platform allows for a remote driver and poses less danger for observers while requiring much less space to run experiments. These benefits far outweigh slightly reduced dynamic similarities and the more limited space and power.

The following sections describe the implementation of the Bluetooth module, the LiDAR sensor, and the wheel encoder's.

Table 1. Bill of Materials (BOM)

<b>Item</b>	<b>Part</b>	<b>Price (USD)</b>	<b>Qty.</b>	<b>Total</b>
1	Traxxas Slash RC Truck	\$279.00	1	\$279.00
2	Slash Roll Cage	\$33.00	1	\$33.00
3	QRE1113 IR Sensors	\$2.95	4	\$11.80
4	6 DOF IMU	\$87.14	1	\$87.14
5	Hokuyo UBG-04LX-F01 LiDAR	\$2,730.00	1	\$2,730.00
6	CMOS Camera Module	\$31.95	1	\$31.95
7	BeagleBone Black (REV B)	\$44.96	1	\$44.96
8	SkyZone 5.8 GHz 200 mW FPV Wireless AV Tx & Rx Set	\$54.99	1	\$54.99
9	Bluetooth Mate Gold	\$64.95	1	\$64.95
10	Bluetooth USB Module Mini	\$10.95	1	\$10.95
11	18650 3.7V Li-ion Batteries	\$8.00	4	\$32.00
12	4x 18650 Battery Holder	\$15.95	1	\$15.95
13	SanDisk Ultra PLUS microSD UHS-1 16GB Memory Card	\$9.99	1	\$9.99
14	Acrylic	\$8.99	1	\$8.99
15	Misc. Hardware and Electronic Components	\$100.00	-	\$100.00
16	California Standard Statewide Tax Rate	Tax	7.50%	\$264.43
				Grand Total \$3,790.10

### 3.5 LiDAR

In accordance with reactive computing and autonomous vehicles, some means of perceiving the environment is required to provide the algorithms with data to process so that an evasive system response may be calculated and performed. There are many methods and sensors that can provide vehicle controllers with the environmental data required to determine an appropriate system response. Radar, LiDAR, vision-based, and sensor fusion approaches have been proposed to provide lane, hazard, and environmental information needed by an autonomous vehicle and active safety systems (Anderson,

Peters, Pilutti, Tseng, & Iagnemma). Light detection and Ranging (LiDAR) sensors use high frequency light waves emitted from a laser diode and detect the time delay or frequency shift of the reflected light wave in order to calculate distances. LiDAR sensors have a very small beam width with less scatter than radar sensors and often use motor mounted mirrors to sweep the beam across a plane.

For this research, a UBG-04LX-F01 Hokuyo LiDAR sensor was implemented to provide the path planning algorithm with the constraint data necessary to find a safe path for the vehicle to navigate as detailed in (Stevens, Carlson, & Painter, Autonomous Collision Avoidance Final Design Report, 2013). The \$2850 sensor provides valid distance data up to 14 feet away every 28.6 milliseconds with an angular resolution of 0.36 degrees. The small footprint and modest power requirements make this sensor well suitable for robotic and scaled RC vehicle applications.



Specification	Value
Horizontal Range	14 ft.
Angular Range	240°
Scan Frequency	35 Hz.
Angular Resolution	0.36°

Figure 39. Hokuyo UBG-04LX-F01 LiDAR sensor

The driver for the LiDAR sensor was provided by the manufacturer supplied sensor library as a C++ library and was implemented into the avoid collisions task. The basic steps to get data from the LiDAR sensor are: make a connection to the sensor, allocate memory, start measurements, receive measured distance data, and close connection. This section summarized the low-level communication with the LiDAR sensor to obtain its data as well as how the data is filtered & formatted to obtain a suitable

point-cloud to pass to the segmentation/parsing algorithm. The next section describes how the steer servo and motor ESC is controlled.

### 3.6 Pulse-Width Modulation (PWM) Motor Control

As discussed earlier, RC vehicles use a radio system with a drive-by-wire interface to decode the pulse-width modulation (PWM) control signals that permit the driver to steer and accelerate the vehicle remotely. PWM is a very common technique to vary the motor's torque and power without using excessive amounts of current (Miller, Ujiie, & Woods, 2011). The principle behind PWM uses a pulsing signal to generate different waveforms and effective voltages that are understood by the motor's ESC and the steering servo. The PWM protocol between the motor's ESC and the steering servo are almost identical. Both PWM control signals run at frequency of 100Hz and vary the duty cycle from 10% to 20% in order to accelerate and steer the vehicle. For control of the steer servo, a duty cycle of 10% yields a steering angle of about 20° to the left and a duty cycle of 18% yields a steering angle of about 20° to the right; a 14% duty cycle produced no steer angle. In regards to control of the ESC, full-reverse is achieved with a 10% duty, full-throttle with a 20% duty cycle, and no throttle (or neutral) with a 15% duty cycle.



Figure 40. Traxxas XL-5 Motor ESC

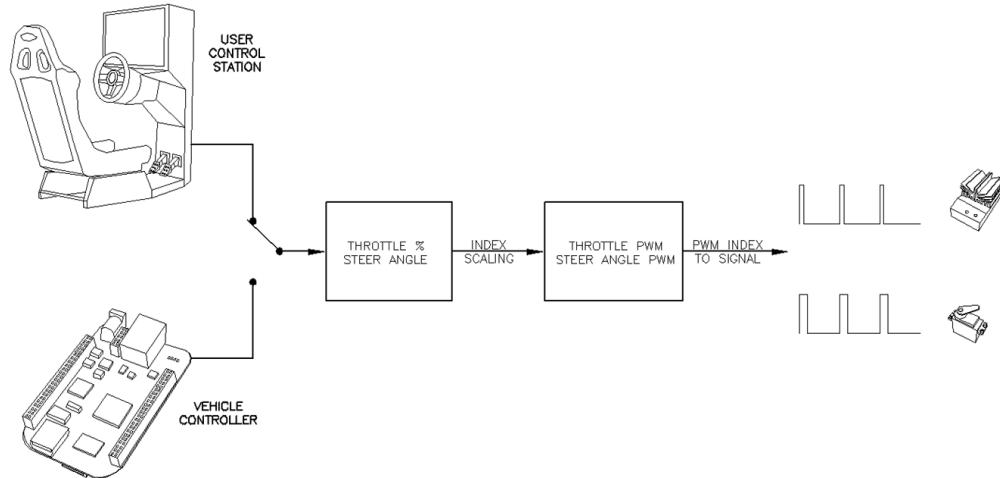


Figure 41. Driver and Controller PWM Scaling

As discussed before, the SoC used on the BeagleBone Black has many hardware blocks and subsystems that allow for easier interfacing with peripherals such as a motor's ESC and a steering servo. One such hardware block is the pulse-width modulation subsystem (PWMSS). Each PWM is a high-resolution programmable module that can support two independent PWM output channels (on a per PWM period basis) (Texas Instruments, 2011). The PWM control signals were generated with the BeagleBone Black SBC by using one of the three PWM hardware blocks built into the SoC. In order to actuate the desired steer angle and throttle, it must be scaled and mapped to a PWM control signal with the appropriate duty cycle. The steer angle is first scaled to calculate the required duty cycle and then this value is mapped to a PWM value that corresponds to a control signal with the required duty cycle. The throttle percentage was mapped to a PWM value between 128 and 255. While this mapping only offers forward acceleration, this also allows this byte to hold values below 128 that are used to signal errors, threat levels, or button presses from the driver's joystick. These PWM values are transmitted over a Bluetooth link as two bytes from the user-control-station to the vehicle.

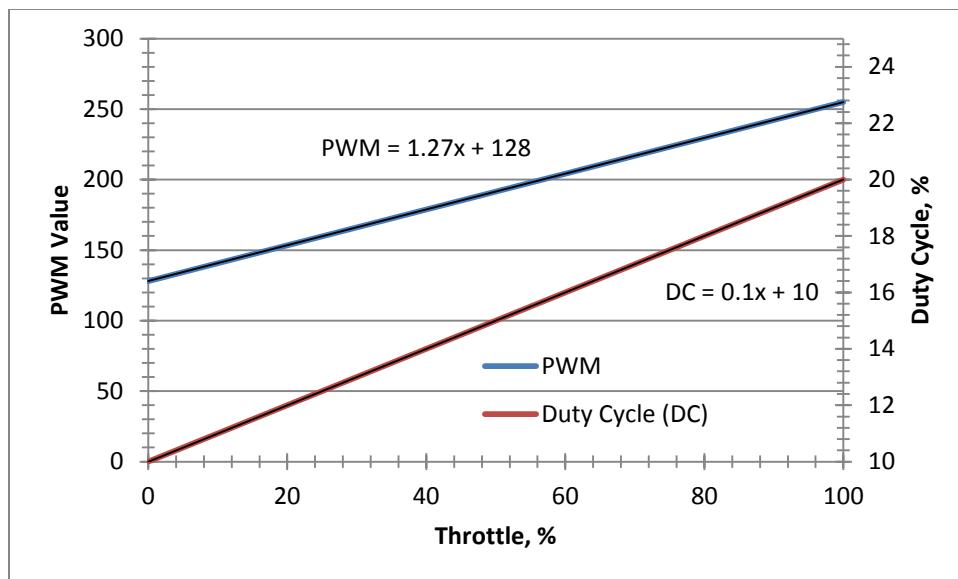


Figure 42. Throttle to PWM Scaling

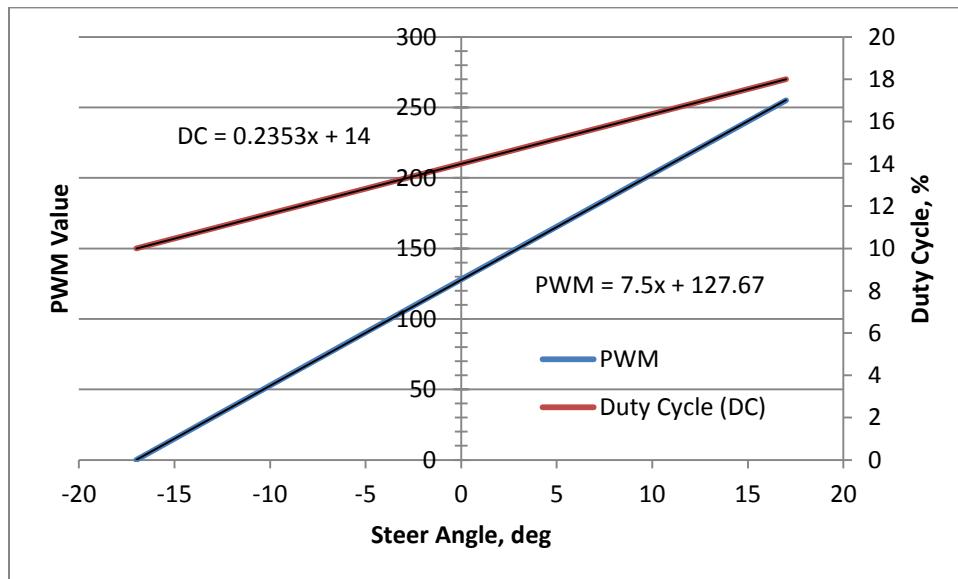


Figure 43. Steer Angle to PWM Scaling

The steer and throttle values may only exist as integer multiples of their respective resolutions, which means the steer and throttle values may only be actuated in discrete multiples. The steering and throttle actuation resolutions may be determined from the scaling above. For the throttle, the actuation resolution is (100 percent throttle)/(255-127) = 0.781 percent throttle/PWM value index. The steering resolution is

$(17 - (-17)) / 255 = 0.133$  deg/PWM value index. Any small error in the steering resolution and calibration is corrected by the feedback loop of the controller. The method used to scale the throttle does not allow for reverse functionality; however, this functionality could be easily incorporated by altering the scaling to allow the driver to switch to reverse via the joystick control interface or may also be incorporated to implement some emergency braking in the case that no safe path can be found.

### 3.7 Bluetooth Serial Communication

In order for the vehicle's embedded single-board computer to communicate with the user-control-station to acquire the driver's control inputs, a Bluetooth link was installed. Bluetooth is a secure standardized protocol for sending and receiving data via a 2.4GHz wireless link; it is an excellent protocol to wirelessly transmit relatively small amounts of data over short ranges less than 100 meters (Sparkfun). Bluetooth networks use a master/slave configuration to control data flow in which a single master device may be connected with up to seven different slave devices. Any slave in the network may only be connected to one single master device. The process of creating a Bluetooth connection between two devices entails that the devices discover each other by sending out an inquiry request and then forming a connection between the devices using the addresses discovered in the inquiry process. By pairing devices together, a bond is formed that allows Bluetooth devices to automatically connect to one another; the pairing process only needs to be performed once. A Bluetooth link is well suited to replace the 100' wired cable that previously delivered the driver's commands to the microcontroller on the vehicle.

Table 2. Power classes of Bluetooth modules

<b>Class Number</b>	<b>Max Output Power (dBm)</b>	<b>Max Output Power (mW)</b>	<b>Max Range</b>
Class 1	20 dBm	100 mW	100 m
Class 2	4 dBm	2.5 mW	10 m
Class 3	0 dBm	1 mW	10 cm

The wireless Bluetooth link in this project uses a low-profile USB Bluetooth adapter on the PC side as the master which connects with the vehicle's Bluetooth module which acts as a slave. For the vehicle side, a Sparkfun Bluetooth Mate Gold board, seen in Figure 44, featuring the RN-41 class 1 Bluetooth module was used to drive the vehicle's Bluetooth communication. The drivers on the PC for Bluetooth communication already exist inherently in Windows. The power output and ranges for each Bluetooth module class can be seen in Table 2. These modems act as (RX/TX) serial pipes that are a great wireless replacement for serial cables and can seamlessly stream data anywhere from 2400 to 115200 bps. The RN-41 class 1 Bluetooth module is a well-documented and easy to use module that offers up to a 350 foot open air range. This provides a robust link and transmission distance while consuming very little power (25mA average). The Bluetooth module operates on the frequency band of 2.402-2.480 GHz. The Bluetooth used to run the experimental vehicle transmits and receive small packets of data at a constant and programmable rate. Each packet contains two bytes; one byte is the throttle PWM and one byte is the steer angle PWM value. With the scaling scheme the range of values for each command can be anticipated and values outside of this range are used to send error codes & button presses encoded within the two bytes alongside the throttle and steer commands. For example, this enables the user to send the shutdown command by

pressing the center button on the steering wheel. Once the two bytes have been received and processed by the vehicle, the average speed read from the encoders is sent as a response to allow the speedometer graphic in the user-control-station GUI to be updated to show the vehicle's speed. The user-control-station and user-control-station's GUI is described below and can be seen in Figure 46.



Figure 44. Sparkfun Bluetooth Mate Gold

### 3.8 Wheel Encoders

As discussed above, a *poll encoder* task/thread is created to monitor each wheel encoder. This task uses interrupts to capture the wheel encoder sensor pulses. Certain pins on the BeagleBone can be configured as interrupt generating based on the value in the edge file and the interrupt controller must be set to poll each interrupt pin. Once configured, the file holding the GPIO state may be monitored by polling. Polling provides a mechanism to monitor a stream for events and an interrupt notifies the computer that the I/O interface is ready to be read or handled. The program returns from the poll function whenever an interrupt is triggered. This requires GPIO filesystem support to allow access to the '/sys/class/gpio' directory. The FSM machine in the *poll*

*encoder* task blocks program execution until the input level on the wheel encoder changes by using the poll method.



Figure 45. QRE1113 Line Sensor

### 3.9 User Control Development and Simulation Interfacing

A user-control-station was developed alongside the embedded controller to relay the driver's steering angle and throttle commands to the vehicle's embedded controller via Bluetooth where the driver's commands are actuated. The user-control station also serves to give a driver's visual perspective. The driver's steering and throttle commands are read within a Java program, running on a PC, from an attached force-feedback joystick featuring standard steering wheel and pedal controls. The vehicle controller sends back the speed and threat encoded in two bytes of data to update the user-control-station's GUI, which includes a speedometer and a threat indicator. A 5.8GHz 200mW video transmitter was mounted on the vehicle to wirelessly transmit the video stream to a video receiver connected to the user control station's television monitor in order to give the user a driver perspective. This transmitter, shown in Figure 48, was selected as it operates on the 5.8GHz frequency band, which helps to avoid interference with the 2.4GHz Bluetooth module. The transmitter also boasts a range of up to 500 meters and permits an operating voltage range of 7-15VDC at 150mA. An omnidirectional clover

leaf antenna set was purchased for both the transmitter and receiver to improve the video signal reception as the vehicle moves and changes orientation. The same CMOS camera was used as implemented previously on the wired system.

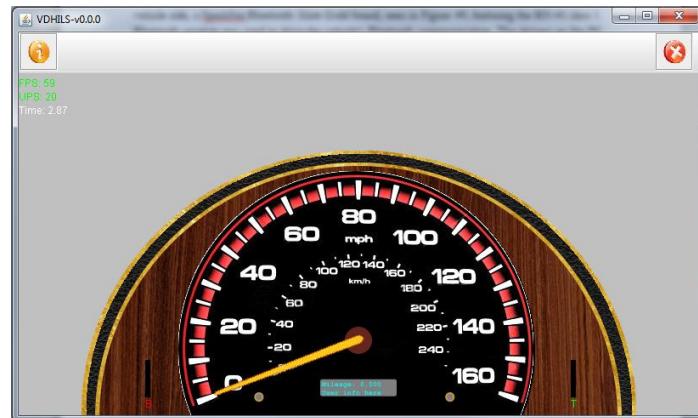


Figure 46. User-control-station and GUI



Figure 47. Logitech joystick with steering wheel and foot pedals

To allow the user to drive the vehicle from a remote PC while utilizing a force-feedback joystick required a user-control-station to be created along with the code to drive it. Incorporation of the force-feedback joystick was made possible by utilizing libraries and APIs, mostly prompted by game development. The force feedback joystick driver for Java that was used was developed as part of a project for the Department of Computational Perception at the Johannes Kepler Universität Linz. This driver provides platform independent support for joystick inputs and for haptic force feedback effects and is based on the Simple Direct Media Layer (SDL) library that provides low level access to audio, keyboard, mouse, joystick, and graphics hardware. In essence, the Java driver is a wrapper for the SDL library through use of the Java Native Interface (JNI) and is compatible with the Light Weight Java Game Library (LWJGL). The driver library provides static methods of a JoystickManager class to obtain Joystick and FFJoystick objects that can be used to obtain input data and play haptic effects. The force feedback effect is used to capture the self-aligning torque and is set to vary linearly with the simulated side slip angle that comes from the dynamic model. This force feedback model

is appropriate assuming that the vehicle's tires operate in the linear region. In this region, the lateral force varies linearly with the slip angle and the vehicle does not slide. The self-aligning torque felt by the driver on the steering column is a function of the lateral force on the front tire multiplied by parameters based on the contact patch and the steering geometry including the trail, the pneumatic trail, and the caster angle. The self-aligning torque provides the return to center functionality of the steering column and must be overcome by the driver (or power steering) in order to steer the vehicle. The gain of the force feedback effect was set through experimentation, although, through further testing and determination of the trail, pneumatic trail, and caster angle a more representative value could be selected. The self-aligning torque is further described by Dixon.



Figure 48. SkyZone 5.8GHz 200mW FPV Wireless AV Tx & Rx Set

### 3.10 BeagleBone Specific Issues and Recompiling the Linux Kernel

As with any single-board computer one can expect there to be some issues and workarounds when using these devices. As more and more users work with these systems under different operating conditions for a variety of applications more fixes and solutions

come to light to remedy these problems. One problem that exists within version 3.8 of the Linux kernel shipped with the BeagleBone is related to the operation of the PWM chips within the System on Chip (SoC). The error does not allow both channels associated with each PWM chip to be used simultaneously due to a bug in the kernel code. As required by the PWM chips, both channels must have the same period; this cannot be done with the default kernel as you can only specify one channel's period at a time, which causes an error because of conflicting channel periods. This prevents the use of both channels. However, by recompiling the Linux kernel this error can be remedied by modifying the source and including some updated firmware. As specified by Saad Ahmad, the PWM kernel driver code was reconfigured to not initialize the channel if it has a period of 0, which allows it to be set later dynamically (Ahmad, 2013). The associated firmware was updated so that each channel had a period of 0 and is disabled when initialized. Then the kernel was recompiled with the updated firmware and installed, which now allows all the possible PWM channels for the Beaglebone to be used.

As discussed earlier, meeting hard real-time constraints within the Linux is difficult as Linux is not a hard real-time OS. Real-time applications are only able to operate correctly if the OS can provide the needed determinism. This entails that higher priority tasks can preempt lower priority tasks. To compound these issues, the BeagleBone Linux kernel has PREEMPT deactivated by default, which would allow tasks with higher priority to preempt tasks with lower priorities. To achieve better time determinism and firmer real-time operation the Linux kernel was recompiled from source with the PREEMPT option activated to reduce random latency.

### 3.11 Running the System and Modes of Operation

Three modes of system operation were created to allow seamless transition between operating the system in open-loop, closed-loop, and closed-loop in a simulated environment. Each mode is accessed and indicated via the UI on the custom BeagleBone Black cape PCB. A description of each mode and how to access it from the cape follows. Depending on which mode of operation was selected, only certain tasks/threads are initialized and started.

#### 3.11.1 Simulation Mode

In simulation mode the embedded controller captures and processes the data stream generated by the real-time simulator. Specifically, the simulator uses a serial FTDI IC to transmit serial data between the vehicle and the PC. The simulator creates an environment similar to the actual test environment including: road cones, obstacle vehicles, and a simulated LiDAR sensor. The simulated LiDAR sensor produces a point-cloud of data that is passed over the serial link to be processed by the controller as if it were running on the ground. After passing the data through the autonomous collision avoidance algorithms, the controller writes the threat level and corrective steer values back to the simulation so that the response may be simulated and observed. To enable the vehicle controller to actuate the system responses to the motors and be able to read the actual wheel encoder sensors that are used to calculate the average speed, a dynamometer platform was devised that allows the vehicle to accelerate and corner while remaining stationary on the desk. An option also exists to either update the speed of the vehicle by either using the measured speed from the wheel encoders or by using the simulated motor dynamics.

### 3.11.2 Free Roam Bluetooth Driver Mode (Open-Loop)

The free roam mode of operation runs the vehicle controller in open-loop control and merely relays the received user's control signals from the PC to the hardware. The speed of the vehicle, read from the wheel encoders, is sent back to the PC so that the speedometer graphic on the GUI may be updated to show the driver the vehicle's speed. This mode of operation starts the following tasks: the motor updater task, the Bluetooth monitor task, the monitor encoders and monitor encoders tasks, and optionally the log data task. If enabled using the appropriate flag, the log data task will record the driver's steer and throttle inputs as well as the measured vehicle speed to a local file on the SBC.

### 3.11.3 Full System Semi-Autonomous Collision Avoidance Mode (Closed-Loop)

The full system semi-autonomous collision avoidance mode runs the closed-loop controller. In this mode the vehicle will remain under user control unless the threat of an imminent collision is determined by the software. The full system collision avoidance software mode begins all tasks discussed above in chapter 3. If enabled using the appropriate flag, the log data task will record the driver's steer and throttle inputs, the measured vehicle speed, and the LiDAR point-cloud to local files on the SBC for later analysis. By recording the point cloud and driver input data, the controllers output can be recreated (i.e. threat, obstacles, road boundaries, and steer angles) in simulation with the ability to enhance and visual the results further.

### 3.11.4 Writing Linux Scripts and Services

For an embedded vehicle controller, it is generally desired that the software automatically starts on boot. Often embedded controllers run as a headless embedded

system without a monitor or graphical interface to initialize, run, and monitor the software. In order to accomplish this, a Linux service can be written to run scripts and programs that setup and run the appropriate software along with a non-graphical user-interface (UI). The Debian operating system used in this project utilizes systemd to manage all system services. Systemd is intended to provide a framework for expressing service dependencies. Creating a system service through systemd allows the collision avoidance software to begin running the code immediately after boot. The service file, ending in “.service,” is added to the “/lib/systemd/system” directory and can be enabled/disabled with the systemctl utility. Once enabled, the system service will start once the single-board computer has booted up.

The service used in this project is described below. The service starts the script called “SoftwareSelectorService.sh.” This script will restart if the software fails to start or runs into an error. The service begins a script that reads the UI button inputs, sets the LEDs on the UI, and starts the software in the selected mode. The software script starts either the open-loop or closed-loop controller depending on which button was pressed.

With the embedded controller software and firmware in place, the process of verifying, tuning, and further developing the system can ensue. As this task can be extensive, time-consuming, and involve much iteration, a method to perform this process without the need to setup and run the vehicle platform on the ground each time a change is made is ideal. A hardware-in-the-loop simulator was developed to aid in this process by allowing the evaluation, tuning, and development of the vehicle’s controller. The next chapter details the design, implementation, and operation of the hardware-in-the-loop simulator.

## **Chapter 4     Developing a Hardware-In-Loop Simulation**

As suggested in (Vahid & Givargis, 1999), simulation is the most common method of testing for correct functionality. Simulation is also used extensively to develop the embedded controller algorithms used in the vast array of electronic control units (ECUs) in modern automobiles to improve performance without relying solely on extensive and time-consuming field-testing. A simulation allows for a much safer environment in which to develop automobiles, their components, and their control systems in order to achieve the desired performance. In addition, a hardware-in-the-loop simulator is a versatile tool to develop, test, and analyze the actual ECU's firmware in a simulated environment. This is extremely valuable for a collision avoidance system. The simulated environment can be tailored to test various scenarios without endangering anyone or anything while running the software on the actual controller hardware. The environment can be set to offer as much space (or lack of) as desired, which may be hard to come by when performing actual field tests. Moreover, complex traffic scenarios may also be repeatedly tested with other static and dynamic obstacles without the need to orchestrate these scenarios in the field. As described in chapter 2, many high-fidelity software simulators, such as CarSim, Adams, and TORCS are available; these packages are generally proprietary packages or the source code is extremely dispersed with little documentation or support. The user is limited to the dynamics implemented in that package and must provide appropriate values for all the parameters used in the model in order to use the software. Identifying all these parameters for each vehicle component and assembly presents another hurdle which takes considerable effort in order to attain realistic and reliable values. For large automotive manufacturers with the resources

and/or infrastructure to perform tests to quantify these parameters, while difficult, does not prohibit the use of these models; the initial effort of quantifying these parameters pays dividends in the end by providing a safe development environment with very realistic dynamic similarity that facilitates rapid development and testing. However, for those outside of the automotive industry it may be prohibitive in regards to cost and time to use one of the high-fidelity modeling software packages and the resources may not be available to identify all the required vehicle parameters. For this project a simple dynamic hardware-in-the-loop simulator was created in order to evaluate and develop the collision avoidance software.

Developing a simulator spans several topics including mathematics, mechanics, physics, and computing. The objectives were to develop a cheap, portable, simple, efficient, and modular hardware-in-the-loop simulator that mirrors the architecture used in the full system by providing a fully immersive driving experience through the use of a force-feedback enabled joystick. The simulator was designed to use the output of the embedded ECU running the exact same collision avoidance software as when run on the ground. A computer simulation with the controller hardware and a human driver in-the-loop was deemed an important aspect of this project to allow the investigation of all the intricacies associated with semi-autonomous collision avoidance systems. Often, the most enlightening observations are made through countless hours of testing and tweaking; the development of such systems that would not be practical with a purely empirical vehicle testing approach alone. This also aligns with the need to ensure stability and robustness of the vehicle controller in all driving conditions and within all dynamic regions; this

exhaustive approach is best described by Eric Raymond, as Linux's Law states, "Given enough eyeballs, all bugs are shallow."

#### 4.1 Simulator Design

To overcome the difficulties of obtaining a proprietary simulator and the associated learning curve, limited interfaces, and the complexities & time required to quantify all of the various parameters for the scaled vehicle, a simple 2D graphical real-time hardware-in-the-loop simulator was developed for this project. The simulator was created to aid in algorithm development for a 1/10<sup>th</sup> scale model collision avoidance system that will be used to study human factors related to this technology and is based on the bicycle model as described in chapter 2. A custom simulator provides more control over the interface and the underlying vehicle dynamics while avoiding the black-box ambiguity of commercial simulators. In addition, the vehicle's sensor could be simulated more easily. The goal while developing the simulator was to strike a balance between complexity and usability that still captures all the important vehicle dynamics most pertinent to developing and testing collision avoidance algorithms. A 2D graphical simulation, developed in the Java programming language, suffices to study these algorithms and human factors as there is no need to use a high-fidelity 3D simulator. Use of pre-existing software libraries were incorporated into this project, when available, to improve productivity and reliability, such as to obtain driver input from the force-feedback joystick. By using the Java programming language and open-source API's the simulator is developed to be platform independent, highly customizable, and easily understandable, especially if future users see a need for change; this creates a very low-cost hardware-in-the-loop simulator for testing and development. A 2D simulation also

drastically reduces the computational overhead when compared to a 3D simulation which makes real-time operation much easier to achieve on average computing hardware; a 2D simulation also lends itself naturally to the incorporation of the bicycle model which ignores the 3D vehicle dynamic effects, including roll and pitch. The human and hardware-in-the-loop simulation runs the same closed-loop software on the same hardware as when the system runs on the ground. This keeps the hardware limitations in perspective throughout development and mitigates the additional effort associated with translating the hardware and software to an experimental scaled vehicle platform for testing. The testing environment can be switched seamlessly. This allows for fast deployment between software updates and gives quick comparisons between the software running in an idealized environment with perfect sensors and to reality with real hardware and sensor constraints. Overall, the simulation reduces the expensive and time-consuming experimental testing and provides verification of the controller before ground testing, which reduces safety concerns, especially in preliminary stages of development, and provides further means to test and design a semi-autonomous collision avoidance system.

As discussed by Milliken, a useful concept when approaching difficult subject matter such as vehicle dynamics is the *Ladder of Abstraction*. This concept applies well to the design considerations associated with the development of a computer simulation designed to capture a vehicle's dynamic behavior, to simulate a vehicle's sensor(s), and to evaluate a controller running ADAS algorithms. The ultimate reality of a vehicle's dynamics is only had by experiencing it and anything less is an abstraction or an approximation of reality; similarly, the ultimate reality of a vehicle's controller can only

be had by using the controller's output. This complete reality is shown at the top and is composed of the functioning vehicle with a human driver behind the wheel and the vehicle's controller intervening when the threat of an imminent collision is detected. The move down the ladder occurs by applying simplifying assumptions, linearizing equations, and simulating sensor & hardware components. The simulation utilizes the bicycle model to simplify the vehicle dynamics to estimate the motions of the vehicle and is suitable for testing autonomous collision avoidance algorithms. The vehicle's embedded controller running the ADAS algorithms is kept in-the-loop by passing it the simulated data and using its output; this moves the results further up the ladder and closer to a complete reality. In addition, a dynamometer was built to allow the vehicle to accelerate and turn while running the simulation on a desk; this also brings the results closer to a complete reality as the actual motor's response and measured wheel speeds can be used within the simulation. The dynamometer is described in chapter 5. The delineation between reality and simulation is shown in Figure 50; the motor dynamics and wheel encoder data is drawn on the line between reality and abstraction as both can either be simulated or measured when the vehicle is used with the dynamometer. A description of how the simulation operates follows.

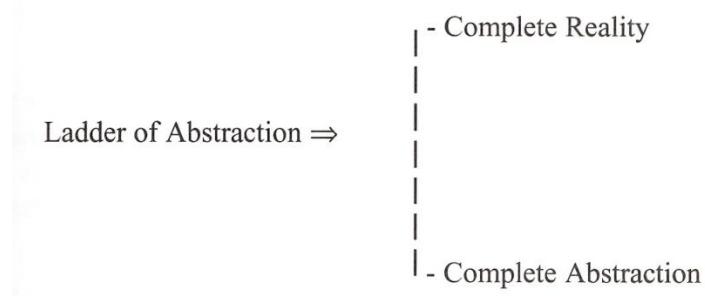


Figure 49. Ladder of Abstraction

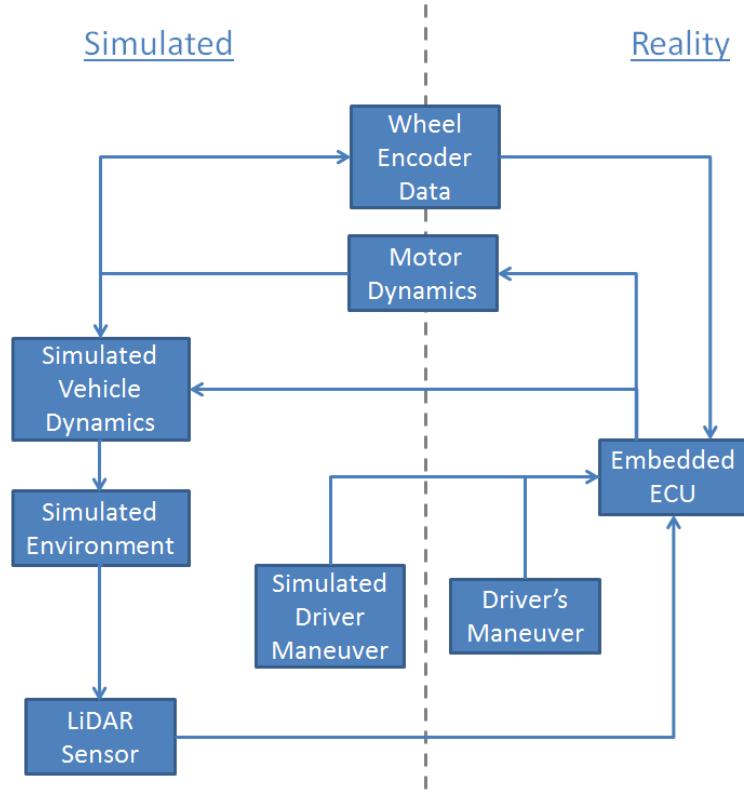


Figure 50. Delineation of Reality and Simulation

The simulation serves to create a virtual environment with obstacles that a simulated LiDAR sensor captures as a point-cloud and that a driver may navigate through with standard steering wheel and pedal inputs. This configuration allows the embedded controller to use the exact same code whether it is actually running on the ground or whether it is running through a simulated environment and provides seamless transition between environments. The simulation passes the simulated LiDAR sensor's point-cloud to the embedded controller via serial communication, which is facilitated by an FTDI breakout board. The embedded controller then operates as if it were on the ground by running the ADAS algorithms and generating a corrective steer response if the threat of an imminent collision is determined. This entails processing the simulated point-cloud by segmenting the data and parsing out road-cones and obstacles. The segmentation results

are then passed through the RRT path-planning algorithm and used to calculate the required steer angle to avoid a collision. The calculated steer angle is passed back to the simulation to update the vehicle's dynamics and to observe the response. The serial connection allows the simulation code and the controller code to be completely separate and modular; only the communication protocol between the simulation and controller must be consistent and predetermined. This allows the development of the simulation and the embedded controller to be completely independent; making changes to the controller's software does not require changes to the simulation nor does it affect the simulation's operation as long as the communication protocol persists.

Just as with the embedded controller's software, a multithreaded architecture was used to run the various tasks necessary to run the real-time simulator. A real-time simulator refers to a computer model of a physical system that runs at the same rate as actual time. Multithreading was incorporated into the design of the simulator to improve efficiency and to meet the real-time simulation constraints. In addition, code was carefully placed to synchronize operation. For example, after the vehicle passes the simulated sensor data to the vehicle's controller it must wait for a response from the controller. While waiting for the vehicle controller to process the data and respond, the simulation performs other tasks associated with updating the vehicle and rendering the graphics. The core of the simulation takes place on three threads: the event dispatch thread, the update thread, and the render thread. The Event Dispatch Thread (EDT) handles all the events associated with the GUI components such as: button presses, checkboxes, radio buttons, etc. The update thread handles updating the simulation physics. The render thread handles drawing the simulation and its objects. Key bindings

were used to process user input from the keyboard. Each class has its own draw and update methods, when applicable, which are called from the render and update thread, respectively. Data is shared across the simulation by using a static data class or by passing references. A core class within the simulation is the car class. With respect to the update thread, this is where the bicycle model based vehicle's physics are implemented, as discussed in chapter 2. This class handles updating all the physics for each instance of the car object. For the driver's vehicle, this class also handles simulating the car's LiDAR sensor scans. The simulated LiDAR sensor captures the point-cloud data of the simulated environment from the car's frame of reference. This is done by sweeping lines across the driving plane and searching for intersections. A line is generated using the Line2D object which originates at the car and spans the sensor's range. The line is searched for intersections with other vehicles and cone objects and the nearest intersection is added as a point to the point-cloud. The simulated sensor's range, resolution, and scan frequency can be adjusted via the GUI and the controller's software permits a point-cloud composed of any number of points. Within the update thread, after a simulated scan the point-cloud data is buffered in the serial communication class and a sub thread is run to pass the simulated LiDAR data to the controller hardware-in-the-loop. The serial communication gets the controller's response and buffers it for the next vehicle physics update. The elegant nature of object-oriented programming (OOP) allows many cars to be simulated simultaneously by creating many instances of the car object that all have the bicycle model based physics and parameters built-in.

In order to run a simulation that constantly updates the physics of each vehicle through numerical integration of the differential equations and then renders the results in

a simulated environment incorporation of a programming loop is required. Game and simulation programming have a lot in common and overlap in numerous ways. Games often simulate the behavior of objects, vehicles, and collisions for the enjoyment of the player. The design patterns between a game and a simulation are very similar. The most quintessential component of this design pattern is the loop, often referred to as a game-loop. This loop is where the physics are updated, the scene is rendered, and where events are processed, such as user input. The game-loop is the heartbeat of every game or simulation. There are many ways to implement a game/simulation loop and the individual implementation of can vary greatly from simulation to simulation (or game to game). The loop dictates when and how fast the simulation state is updated and the scene is rendered. A simple loop may have the form below in Figure 51.



Figure 51. Simple game-loop/simulation-loop

The above loop has no timing considerations and operates the rendering and physics updates in the same loop. However, both the simulation physics and rendering loop need to run at different rates while regulating the time. This is accomplished using multiple threads, one to run the simulation update loop and one to run the rendering loop. Each loop regulates its speed by using a high resolution system timer. The simulation

must be updated much faster than the scene needs to be rendered in order to maintain numeric stability regarding the vehicle physics. In addition, rendering faster than the monitor's refresh rate or faster than is perceivable by the human eye is unnecessary and incurs high processor overhead. Real-time operation is realized by updating the simulation and scene with the actual elapsed system time. An example of a time regulated loop is shown below in Figure 52.

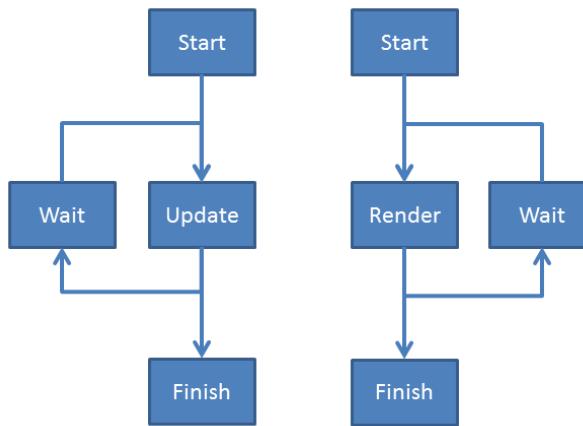


Figure 52. Multithreaded game-loop/simulation-loop with time regulation

The use of a game loop allows for complete control over the timing of simulation and for active rendering. This is of the utmost important, especially in game and real-time simulation development, as the distances and state of the vehicle sprite have a physical dependence on the elapsed time.

#### 4.2 Motor Dynamic Simulation and Parameters

A linear motor model was adopted to simulate the motor's response if the actual encoder speeds are not used in conjunction with the dynamometer. This rudimentary model was implemented into the simulation to provide engine characteristic curves. The linear motor model parameters can be measured and evaluated from manufacturer data. While this is not the most accurate model, it has very little computational overhead.

Evaluating the parameters for the motor and incorporating a motor to drive wheel gear ratio allows estimation of the drive torque and the propulsive tractive force. By referring to the manufacturer's data sheet for the Titan 12T 550 brushless motor we find that the motor has a nominal no-load speed of 50,000 RPM and a Kv value of 3500 RPM/V. For use in the simulation, the throttle pedal needs to modulate the motor power at any given operating speed. This was accomplished by shifting the power curve up and down by adjusting the y-intercept value. The resulting curves can be seen in Figure 53.

Mathematically, this can be described by the equation below:

$$T_m = \frac{-K_b K_t}{R_a} \omega_m + \frac{K_t}{R_a} E_a (\% \text{ THROTTLE}) \quad (32)$$

Some power curves are shown below for a simple DC motor.

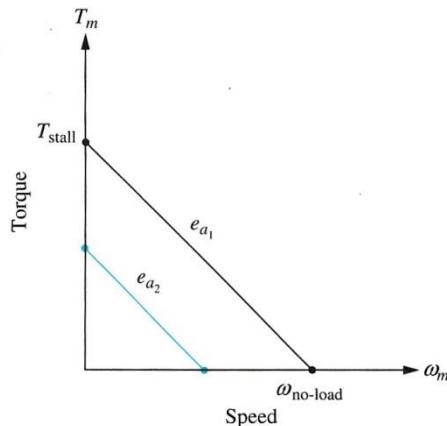


Figure 53. DC Motor Model Power Curves

Also described in chapter 2, low speeds cause estimations of the slip angles to approach infinity and a low-speed model must be incorporated to maintain stability. There is a parameter in the simulation, adjustable from the GUI, that controls the speed at which the simulation switches from using the low-speed to the high-speed model. The value of this parameter is set for smooth transition and to avoid introduction of model instability. The value of this parameter was observed to depend on the vehicle being simulated.

#### 4.3 Vehicle Model Interface with Embedded Controller

To decrease the transmission time between the simulation and the controller in the loop each float value within the point cloud data was scaled by 100 and then cast to be represented by a short, or 2 bytes. This method retains two place decimal accuracy for the values in the LiDAR sensor's point-cloud. The throttle and steer outputs are single unsigned bytes that have been scaled from desired steer angles to the corresponding duty cycles and finally a PWM value that maps the duty cycle range to a byte (or 0-255). The serial communication link between the embedded controller and the simulator allow for completely different software architecture and programming languages to be used. Both the embedded controller's software and the simulator's software can be modified independently without affecting each other's operation. The simulator is written entirely in Java and the embedded controller code is written entirely in C++. The embedded controller captures the serial data generated by the simulator and runs the simulated data through the autonomous collision avoidance software algorithms and spits back the output to simulate the vehicle's response.

#### 4.4 Creating and Rendering a Simulated Environment

The roadway is created within the roadway class and was implemented to provide an endless, scalable driving environment that can be customized to the user's preferences. The roadway is created using road tiles that connect together using a rule that a roadway must connect to another roadway. With all the roads ending in the center of one side of the road tile, seamlessly endless maps of any variety can be created. Enough tiles are used to fully populate the screen. When the user's vehicle nears the edges of the screen the roadway tiles move instead of the vehicle to prevent the car from traveling off of the

screen. The camera can be repositioned and panned left, right, up, or down to reposition the car within the viewport. The images below show some typical road tiles as well as some example roadway scenes.

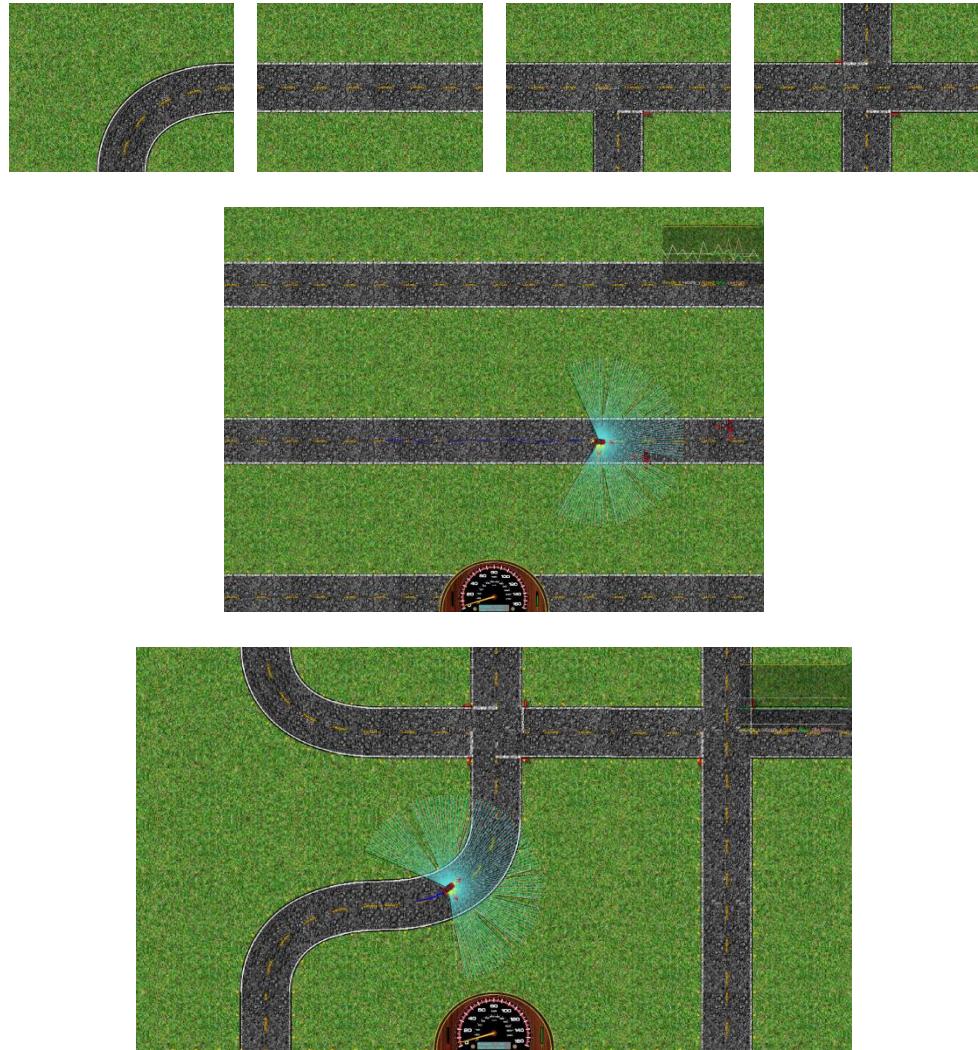


Figure 54. Road tiles and example generated road maps

The various coordinate systems used in this project required that coordinate transformations be performed when using data between reference frames. The coordinate system used in computer graphics has an origin in the upper left corner of the screen with the positive x-axis oriented across the screen and the positive y-axis oriented down the screen. The coordinate system for the simulation is situated in the middle of the screen,

initially, with the positive x-axis oriented to the right in the positive sense and the positive y-axis oriented up. The origin of the simulation may change with respect to the coordinates on the screen as the viewport translates when the car nears the edge of the screen or the user pans the camera. The viewport may also be scaled to facilitate zooming in and out. The viewport translation and scaling is independent of the physics and only the rendered viewport scene is scaled. This is accomplished by establishing a pixels per meter value for scaling which may be changed to zoom in and out of the rendered scene. This scaling value allows mapping between the representative rendered scene in pixels and the physical environment in meters. In addition, a vehicle coordinate system is fixed to each vehicle and rotates and translates as the vehicle moves. The algorithms used to simulate the LiDAR sensor and to process the data have different reference frames and the appropriate transformations must be applied to yield meaningful results.

#### 4.5 GUI, User Control Development, and Simulation Interfacing

The steering and throttle commands from the driver are read into the simulation in the same way as in the user-control-station, described in chapter 3. Before the steering angle read from the driver's steering wheel is used to update the vehicle's physics the value is scaled by the steering ratio. This is the ratio between the angle of the steering wheel and the angle of the vehicle's wheels. In most passenger vehicles, this ratio varies between 12:1 and 20:1. The simulation allows this value to be adjusted to control the steering sensitivity perceived by the driver. If no joystick is used or detected, the driver can use the keyboard to control the vehicle. The keyboard controls are shown in and allow the user to operate the simulation and may be used to drive the vehicle. The vehicle

can also be controlled automatically by specifying the desired maneuver and speed as described in the next section.

#### 4.6 PID Speed Controller and Primitive Vehicle AI

Primitive driver artificial intelligence (AI) was implemented by using the geometric Ackermann steering based pure-pursuit control law model, as described in chapter 2. The control law allows the required steer angle in order to follow a specified path to be calculated. The user may specify typical driving maneuvers similar to those found in CarSim including: a Single-Lane Change (SLC), a Double-Lane Change (DLC), Center, or None. Each vehicle's maneuver may be set and enabled from the GUI by the user. Upon starting a new simulation, a path is generated for each obstacle according to the selected maneuver and the surrounding road tiles. The process is illustrated below in Figure 55 for a SLC maneuver.

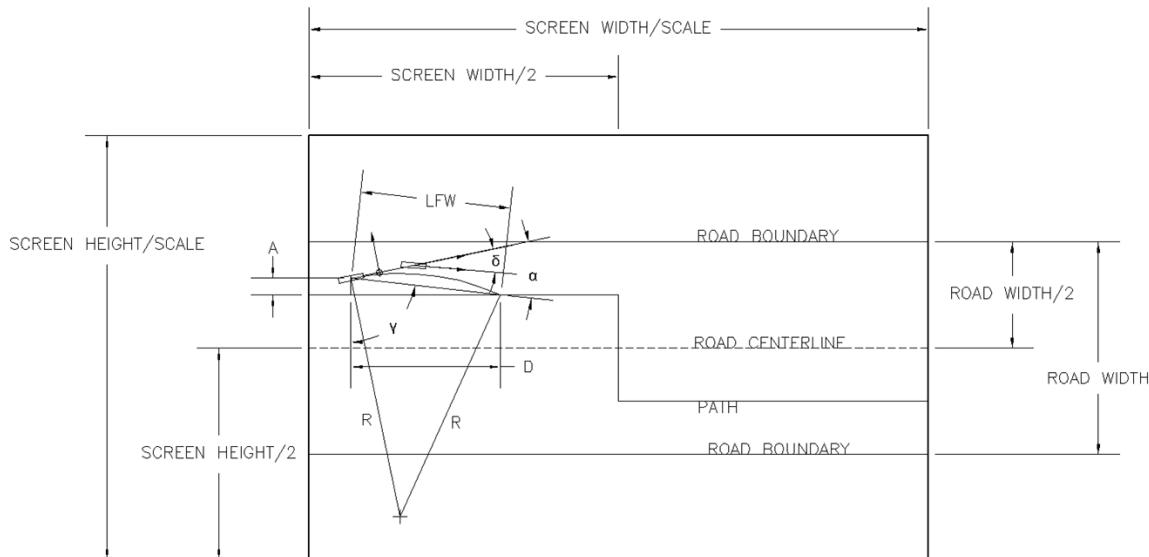


Figure 55. Example Path Generation for an SLC Maneuver

$$\gamma = \cos^{-1} \left( \frac{A}{LFW} \right) \quad (33)$$

$$\alpha = \text{Car Angle} - \text{Angle of Path} + (180 - \gamma) \quad (34)$$

In order to simulate the driver maintaining the speed of the vehicle during the AI maneuver, a proportional-integral-derivative controller was used to accelerate the vehicle and to maintain the target speed. The gains can be set to emulate different driving styles and the aggressiveness of the driver on the controls. This can be used in conjunction with the driving maneuver AI to allow consistent simulation of specific maneuvers of the drivers' or obstacle vehicles. Driving AI is one feature of the simulation and a summary of some other features are listed below:

- Simulated LiDAR sensor with variable resolution and scan frequency
- Real-Time Signal Plotter
- Configurable AI maneuvers for both user and obstacle vehicles
- Configurable PID speed controller for both user and obstacle vehicles
- Joystick or keyboard control
- Adjustable vehicle bicycle model parameters from GUI
- Add/remove obstacles and set their parameters from GUI
- Save/load current simulation
- Operation in passive (without ECU) or active (with ECU) mode
- Measure distances
- Rendering of force vectors acting on each vehicle
- Speedometer

With the embedded controller developed and implemented on the vehicle, which may be operated from within a hardware-in-the-loop simulator, the testing results follow in the next chapter.

## **Chapter 5 Testing and Results**

This chapter describes the test results of the embedded ECU controller, the dynamometer, and the hardware-in-the-loop simulator. Additional tests were conducted in order to determine the vehicle's parameters for the simulator's vehicle model and are also detailed. An instance of how the simulator can be used to develop and improve the controller's performance is presented by altering the threat determination algorithm.

Also, an instance of how the simulation can be used to identify issues with the controller's software is shown. A post processing tool is presented and is used in conjunction with the simulator to aid in the analysis and development of the controller's software. This post processing tool facilitates the evaluation and visualization of the controller's algorithms by creating a video, plots, and a table that summarizes each autonomous controller override event. This tool eases the development and tuning of the controller by allowing the algorithms to be modified, tuned, and then rerun against the same data. The degree of improvement between the modified and the unmodified algorithms may then be evaluated by comparing the output from the post processing tool. The next section describes the tire tests performed to characterize the tire's properties and the high-fidelity solid model of the vehicle used to determine its mass properties. These parameters are required for use with the simulator's bicycle vehicle model.

### **5.1 Tire Tests and Tire Testing Machine**

Many methods exist to measure and determine the parameters that characterize a vehicle. Some of these methods are much better suited for measuring scaled vehicle properties. For this project, a high resolution SolidWorks model was developed. The SolidWorks model permits estimation of the vehicle's mass and geometric properties.

Additional tests were performed in order to determine the tire's properties. A summary of the values used to simulate the scaled vehicle are shown below in Table 3.

Table 3. Parameters of Traxxas RC truck

Parameter	Symbol	Value
Sideslip Coefficient of Front Tire	$C_{\alpha f}$	63.2 N/rad
Sideslip Coefficient of Rear Tire	$C_{\alpha r}$	63.2 N/rad
Vehicle mass	m	3.117 kg
Moment of Inertia	$I_z$	0.0452 kg*m <sup>2</sup>
Distance from C.G. to Front Tire	a	0.212 m
Distance from C.G. to Rear Tire	b	0.126 m
Pacejka tire model peak factor	D	70 N
Pacejka tire model shape factor	C	0.09
Pacejka tire model stiffness factor	B	82
Pacejka tire model curvature factor	E	0.65

As discussed in chapter 2, the tires play a central role in determining the dynamic behavior of a vehicle and its motions. Tests were conducted to determine the tire parameters for the *Magic Formula*. In order to use an empirical tire model to estimate the forces developed by the tires, data must be obtained and reduced to find the value of each parameter in the model. The parameters that appear in the formula represent the physical characteristics of the tire. A skid-pad test was performed as a simple means of relating the sideslip angle to the amount of force generated by the tires. This test was conducted by driving at a constant speed in a circular motion with a constant steer angle while monitoring the speed and radius of the circle circumscribed by the vehicle. The wheel speeds were monitored with wheel encoders and the radius was measured in the field with a tape measure while the vehicle performed the test. The side slip angle was estimated by subtracting the measured steer angle by the ideal no-slip Ackermann steer angle for the observed radius at the measured speed. The lateral force was estimated with the measured steady-state velocity, radius, and the vehicle's mass. To estimate the lateral

force provided by each tire during the skid-pad test, the total lateral force was averaged over the four tires by imposing a neutral steer assumption, which means the slip angle was assumed to be equal for the front and rear tires. Further tests were performed by creating a tire dynamometer. A tire dynamometer was assembled by retrofitting a small wood lathe for the purpose of testing small tires in order to characterize the tire's ability to grip and generate lateral cornering forces. This also provided an estimation of the tire's cornering stiffness and helped to validate the skid-pad test results. Various springs were used to measure the force generated by the tire that was affixed to a carriage. First, the springs used to measure the force were characterized to find their respective stiffness's. Different springs offered different resolutions and capacities when measuring the lateral force generated by the tire. The tire was set and locked at different angles. The lathe was then started and the carriage holding the tire was allowed to travel down the ways until steady-state was observed, which occurred when the tire's lateral force and the spring force came into equilibrium. Once steady-state was achieved, the spring's deflection was recorded and the corresponding force was calculated. This test was performed at a variety of slip angles with varying vertical loads in order to generate the characteristic tire curves. These tests facilitated estimation of the tire's cornering stiffness. The tire dynamometer and skid-pad tests are shown in Figure 56.

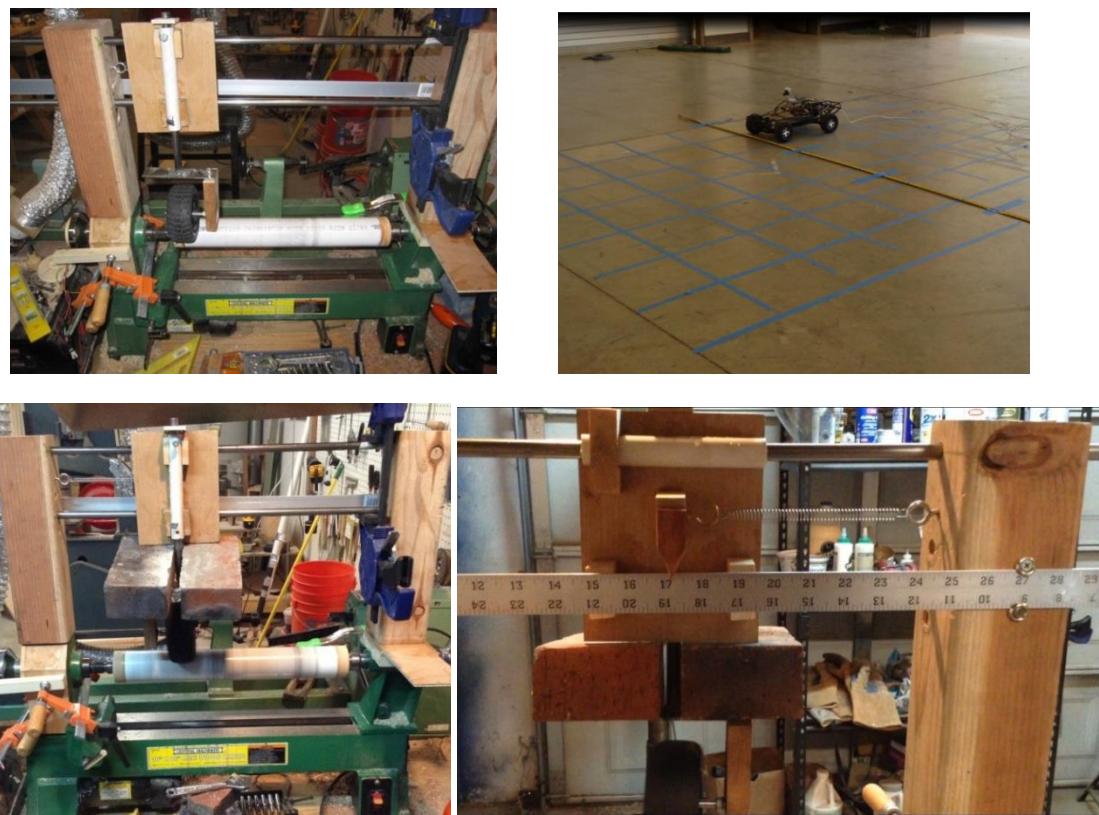


Figure 56. Skid-pad (upper right) and tire dynamometer tests (upper left, lower left, lower right)

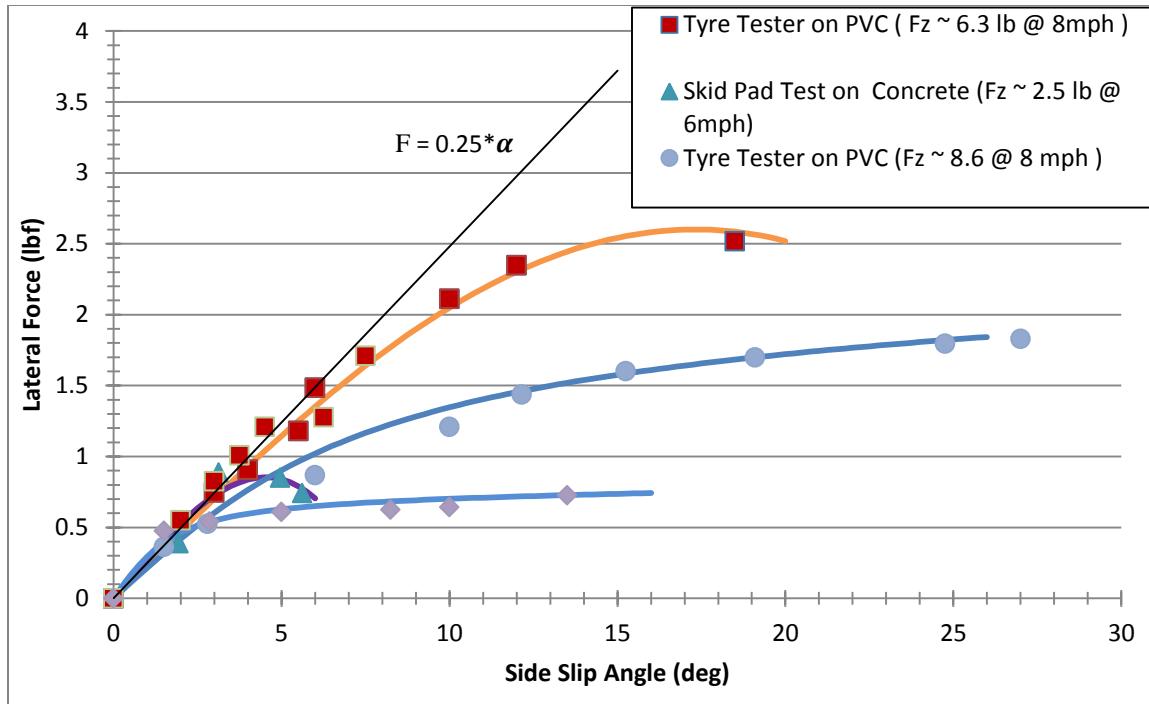


Figure 57. Tire dynamometer and skid-pad test results performed with varying vertical loads ( $F_z$ )

The plot in Figure 57 shows the force generated per tire at a given sideslip angle. The Pacejka *Magic Formula* tire model curve was fit to each data set. The effect of different vertical loads was observed by altering the load and observing the change in the shape of the curve. The greater the vertical load the greater the adhesion between the tire and the road surface. Increasing the vertical load increased the peak lateral force and retarded its occurrence with respect to the slip angle. The effect of different coefficients of friction was also observed by comparing the smooth, polished concrete skid-pad test results to the tire dynamometer test results performed on a roughened PVC surface. A higher coefficient of friction also increased the peak lateral force and retarded its occurrence with respect to the slip angle. A tire's cornering stiffness is determined by evaluating the slope of the characteristic curve at the origin, as expressed in Equation 35.

$$C_\alpha = \left( \frac{\partial F_y}{\partial \alpha} \right)_{\alpha=0} \quad (35)$$

The tire's cornering stiffness was estimated by evaluating the slope in the linear region of each curve nearest to the origin. In accordance with Equation 35, the slope of the linear region for each curve was estimated using the origin and the first data point. This yielded similar estimates of the cornering stiffness for each trial. The tire's cornering stiffness was estimated to be  $0.25 \pm 0.05$  lbf/deg. These tests only provided a rough estimate of the non-pneumatic tire's cornering stiffness, largely due to the small sample size and the propagated measurement errors. However, performing these tests was more favorable than basing a value on scaled pneumatic tire research alone, creating a more intricate tire dynamometer, or performing a very complex finite element analysis of the tire. For comparison, typical tire dynamometer tests measure around 15 parameters to fully characterize the lateral tire dynamics and to evaluate the lateral Pacejka tire model parameters. In the next section, the dynamometer that may be used with the simulation is described.

## 5.2 Vehicle Dynamometer

A vehicle dynamometer was constructed with the assistance of an undergraduate student that permits the actuation of the steering servo and the brushless motor while running the hardware-in-the-loop simulator. This allows for utilization of the vehicle's actual wheel encoder data during simulation. The vehicle is fixed onto the dynamometer with acrylic brackets that hug the frame in place while the vehicle's front and rear wheels rest on rollers that are free to rotate. A belt is placed around the front and rear roller. The torque delivered by the motor through the rear wheels and onto the rear roller is transmitted to the front roller through the belt so that all wheels rotate together as if the

vehicle were traveling along the ground. The rollers were selected in order to match the inertia of the vehicle accelerating along the ground. By allowing the speed to be measured from the wheel encoders, the differential equation used to estimate the speed of the vehicle and the linear motor model used to estimate the motor's torque at each time-step in the simulation are not necessary while the vehicle operates on the dynamometer platform. However, since the speed measured from the encoders is a scalar the sideslip angle still needs to be simulated in order to break the measured velocity into vector components. The resulting tests with the dynamometer had a similar motor response and wheel speeds as when the vehicle operates on the ground. Unfortunately, the belt used to transmit the motor's torque from the rear axle to the front axle often binds and jams with adverse effects. Another iteration of the dynamometer is required so that it can be used with the simulation without hindering the vehicle's dynamics or degrading the simulated results. The use of the simulation with the dynamometer was forgone until this issue can be addressed; however, the software is in place and can be enabled via the GUI to permit future use. The next section describes results from the hardware-in-the-loop simulator and some typical use cases.

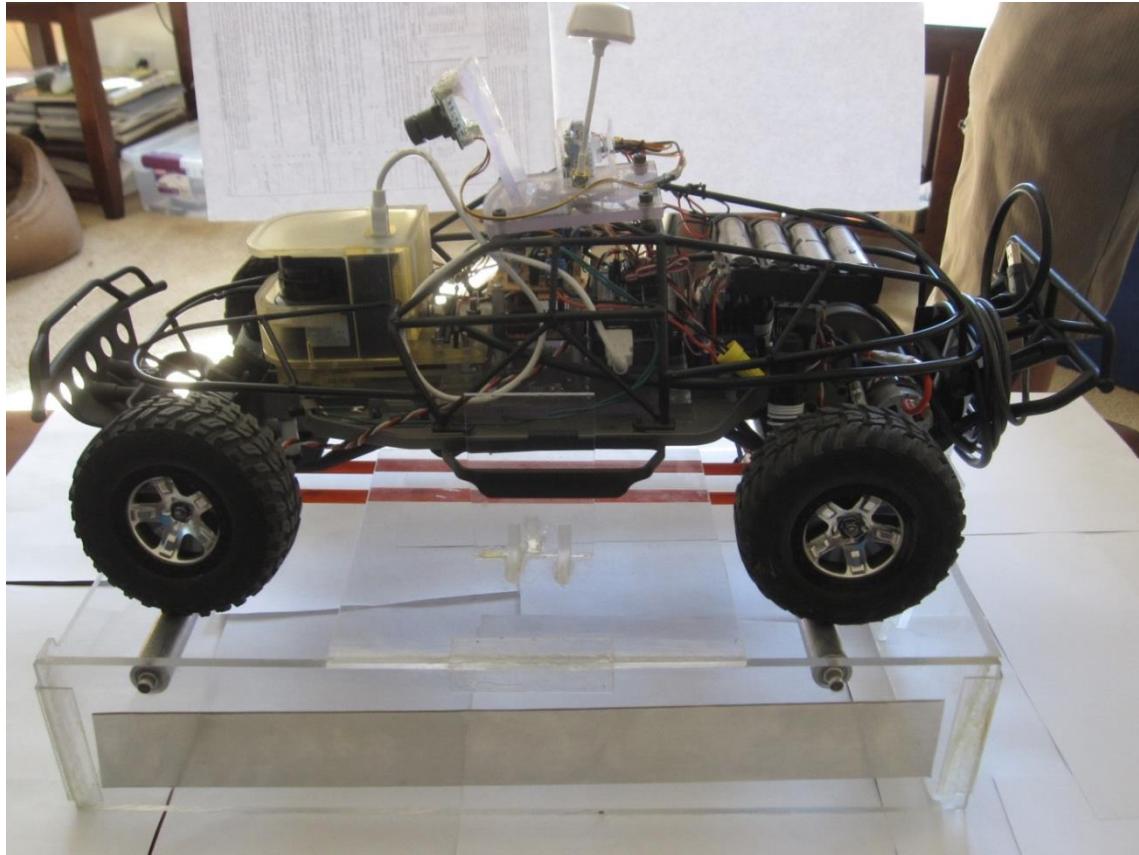


Figure 58. Vehicle dynamometer used in conjunction with the simulation

### 5.3 Simulator Post Analysis Tool

A post analysis tool was created to allow the data collected from the simulation or in field tests, in active or passive mode, to be visualized, analyzed, and evaluated. Specifically, this tool facilitates visualization and analysis of the controller's underlying segmentation and path-planning algorithms that predominately affect its performance. The post analysis tool generates a video, plots, and a table of statistics for each controller override event. This tool was written using open-source libraries and applications, such as Gnuplot, OpenCV, and FFmpeg and is platform independent. It was created as a way to process the data from the simulation in order to evaluate the controller's response with an iterative capability. A description of how the tool functions follows. First, the tool reads

in the point-cloud data and the vehicle data recorded from the simulator. The vehicle data includes the steer angle, the throttle position, the brake position, the vehicle position in the global frame, the velocity, the yaw angle, the yaw rate, and the vehicle controller's response during the simulation. The point-cloud data and the vehicle data are synchronized with the use of timestamps. The tool then runs the segmentation and path-planning algorithms, which may be modified and tuned even after data is collected. This is valuable even if the point-cloud does not change in response to a modification of the software algorithms because each point-cloud is considered independently. This tool could be further developed to compensate the collected point-cloud data to track the difference in the vehicle's trajectory between the modified and the unmodified software by using the dynamic model. The original controller's responses executed during simulation and the altered controller's responses with the modified/tuned algorithms are plotted together enabling a side-by-side comparison. For each point-cloud in the collected data an image is created that plots the results from the modified/tuned segmentation algorithm, the threat algorithm, and the path planning algorithm on top of the LiDAR's point-cloud. The results of the controller's response during the simulation are overlaid on the left side of the image to show the controller's threat level, the detected yaw angle, and the issued steer angle before any modifications or tuning of the algorithms. The right side of the image shows the same information for the controller's response with the modified controller algorithms. An example of this image is shown below in Figure 59. All the output images are merged together using FFmpeg to create a video. An analysis of each controller override for the modified/tuned controller algorithms is generated, which corresponds to the rising edge of the threat signal. This includes the duration and distance

of each override, the number of RRT iterations performed, the number of obstacles, the number of threatening obstacles, the threat region, the threat trigger (obstacle, road bounds), and a link to the output image that corresponds to the rising edge of the threat signal. Plots of the unmodified and modified controller algorithm outputs are plotted, including the threat signal and the steer angle. In addition, plots of the vehicle's signals are produced, such as the speed, the yaw, the yaw rate, the throttle, and the brake. Together, the video, table, and plots that compare the unmodified and modified controller algorithms output are extremely useful in determining if the alterations improved the controller's performance. This drastically helps to drive development and ease tuning of the controller. This tool allows modifications to be made to the segmentation, threat, and RRT algorithms, rerun with the same data, and compared with the controller's original response, which eases controller development and tuning. This tool is very useful to observe how the controller's software processes and responds to an imminent collision or an imminent departure from the roadway. The post analysis tool allowed the software to be altered and the same data to be analyzed facilitating comparison of the ECU's response with the simulated ECU's response. This allowed a targeted and iterative way to further refine the threat determination that lead to a reduction of unjustified controller override events and rapid switching between driver and computer control by about 90%. This metric was determined by analyzing the output from the post analysis tool with data from a few simple scenarios: one scenario with no obstacles and one scenario with an obstacle in the center of the road way. The number of rising edges of the threat signal were then counted in each case for the original software and the modified software. The unjustified oscillations between driver and computer control for the modified threat

determination algorithm was significantly reduced. This tool was also used to investigate implementation of a braking mechanism if a justified threat is determined but the RRT fails to plan a valid path in time. By braking in proportion to the estimated time to collision, the vehicle can slow down to mitigate damage and to allow the controller's path-planning algorithm more time to find a safe path in which to follow. This simulated brake assist functionality is shown in Figure 60. This feature was not tested on the vehicle hardware but shows how such a feature may be first developed in simulation.

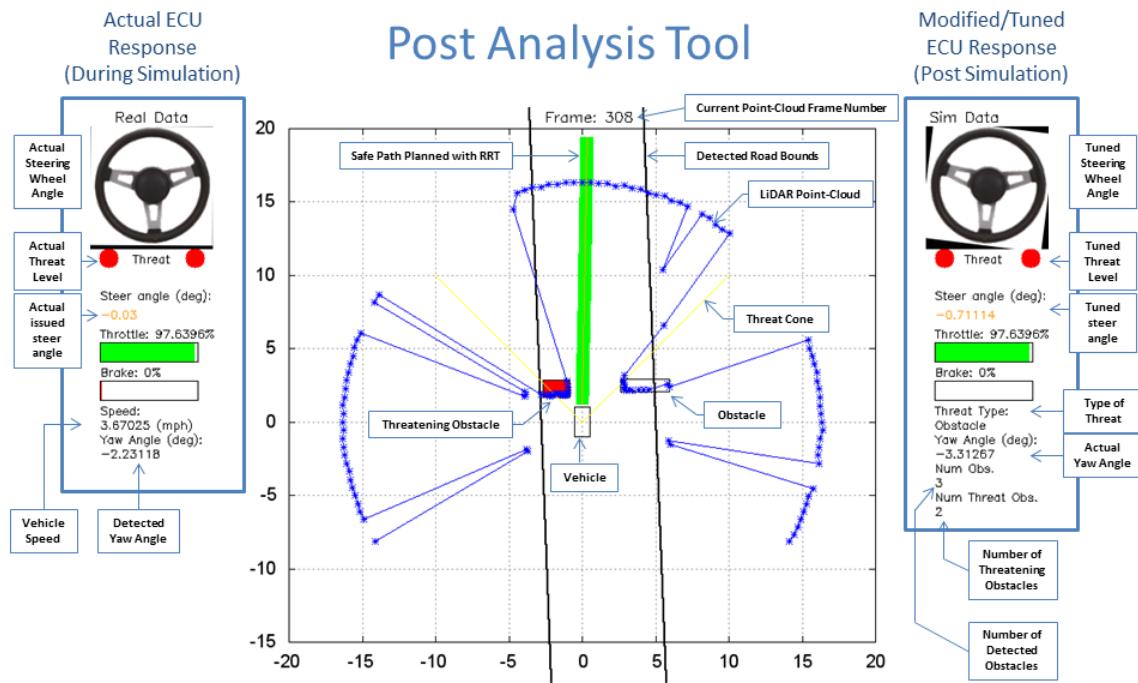


Figure 59. Example of post analysis tool's output

Figure 59 above shows a representative image of one point-cloud processed through the post analysis tool. This image helps to see how the controller is processing the data and reacting to each point-cloud across software revisions. Each algorithm's output from the point-cloud data is shown including, the segmentation, the threat determination, the path-planning and output steer angle. The center of the image shows

the original LiDAR's point-cloud, plotted in dark blue, on a scale with units equivalent to those that compose the point-cloud data. The vehicle is plotted as an unfilled black rectangle centered at the origin. The solid black parallel lines show the detected roadway, slightly offset from the detected cone obstacles towards the vehicle. Detected obstacles are plotted in the same manner as the vehicle, as unfilled black rectangles. Obstacles that fall outside of the detected road boundaries are removed from the set of detected obstacles. Obstacles within the roadway and within the threat cone, shown as two yellow angled lines emanating from the center of the vehicle, may be deemed threatening by the controller. Threatening obstacles are plotted as red filled rectangles. If a threat exists, the safe path determined from the RRT algorithm is plotted as a filled green contour. All paths that fail to be planned within the RRT algorithm are plotted as cyan contours. On the left side of the image, the ECU's response while data was collected is shown. This includes the threat level which is shown as a red filled or green filled circle corresponding to a high threat level and a low threat level, respectively. The actual steer angle is shown, numerically, as the angle of the steerable front wheels and, visually, as the angle of the steering wheel. If the threat is high, the steering wheel angle is given by the output from the controller and actuated through haptic force feedback effects; otherwise, this is the steer angle issued by the driver. Similarly, the brake and throttle percentage is shown from the driver if the threat is low or from the controller if the threat is high. The speed of the vehicle, determined from the vehicle dynamic model, is shown in miles per hour. Also, the detected yaw angle of the parallel road lines, determined from the segmentation algorithm, is shown. On the right side of the image, the modified controller's steer response, threat level, and throttle and brake percentage is shown in the

same manner. In addition, the number of detected obstacles and the number of threatening obstacles is shown for the modified controller's software. The type of threat is also listed and its color changes according to the threat region.

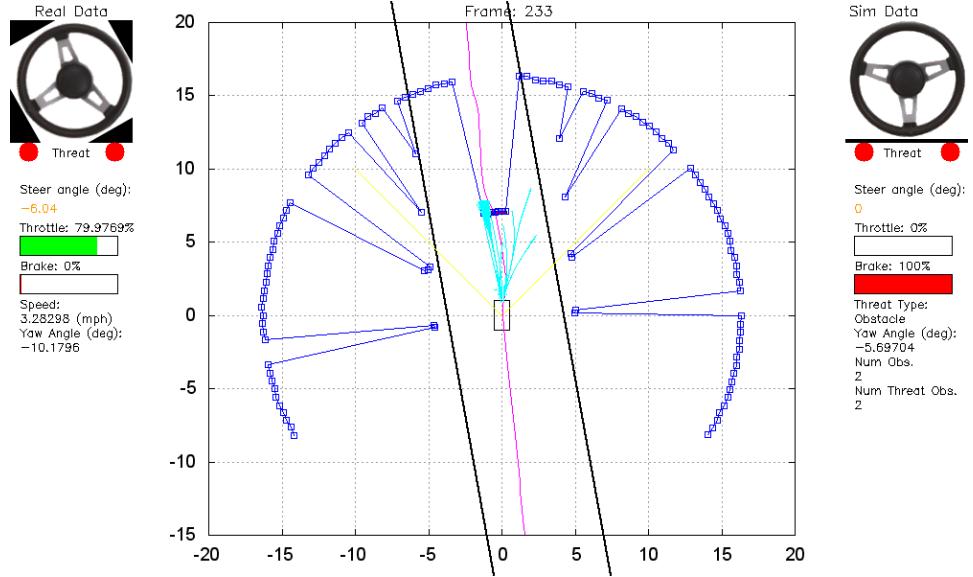


Figure 60. Brake assistance functionality when no safe path can be found and an imminent threat exists (cyan curves show failed paths, no valid paths present, brake percentage shown in left panel)

In Figure 60, the simulated brake assist functionality is shown. The cyan contours show all the failed attempts to find a safe path around the obstacle in time while running the path-planning algorithm. Due to the random nature of the RRT algorithm, it can be seen that during simulation the controller in the loop did successfully find a safe path and steered the vehicle to the right while in simulation no safe path could be found. By braking in relation to the estimated time to collision, more iterations and attempts to find a safe path can be run before reaching the obstacle or at the very least damage can be prevented or mitigated. The braking event shown calls for 100 percent braking to mitigate damage and to allow for more time to find a safe path. The general workflow of the

development toolchain, which consists of the hardware-in-the-loop simulation and the post analysis tool, to further develop the controller's software and to analyze problems is as follows: first simulate the scenario with the hardware-in-the-loop simulator and the existing controller software, next analyze and evaluate the simulated data and the controller's response with the post processing tool, finally modify the controller's software and re-evaluate by comparing the new response to the original response through use of the post analysis tool.

#### 5.4 Hardware-in-the-Loop Simulator Testing

The hardware-in-the-loop simulation is a versatile tool that can be used to debug, develop, and evaluate the embedded ECU's software as well as the effectiveness of semi-autonomous, steering-based collision avoidance systems. As described in chapter 4, the simulated environment and LiDAR sensor data is fed into the ECU in the loop and the response is used to simulate the vehicle's dynamic response. Some examples of how the simulation and post analysis tool can be used to investigate the controller's effectiveness follow below. The specific issues investigated were the effect of the LiDAR sensor's resolution on the controller's performance, the sensitivity of the threat determination with respect to the road boundaries, and reducing the oscillations in the output of the RRT algorithm by following the same path. The next section describes the ESV algorithm used in the prior wired implementation of this project.

##### 5.4.1 Previous ESV Algorithm

The ESV algorithm implemented on the previous wired system that ran remotely on a PC within a MATLAB Simulink model showed much strength but was not without its weaknesses. The benefits included real-time operation, effective segmentation and

road boundary determination from the LiDAR's point-cloud data, and effective path-planning. The LiDAR sensor based obstacle and road boundary detection worked extremely well and will continue to be an integral part of this platform in the future. In addition, alternative LiDAR sensors with a lower price and lower resolution may also come to market and may be investigated for use in this system. The previous implementation was successful in producing the desired semi-autonomous collision avoidance behavior and allowed drivers to experience a scaled version of this technology, which facilitated a preliminary human factors study to be performed. The weaknesses of the previous implementation became more apparent when the wires were removed, especially concerning the determination of the threat and handling the switch between user and computer control. Some of the weaknesses of this algorithm included classification of obstacles as threatening or nonthreatening, the oscillatory nature of the RRT algorithm's output especially when faced with a single object in the center of the roadway, consistent detection of the road boundaries, and the estimation of each detected obstacle's velocity. The next section describes how the simulation was used to investigate the LiDAR sensor's resolution on the controller's performance.

#### 5.4.2 LiDAR's Resolution Influence on Controller Performance

The simulation can be used to investigate how the sensor impacts the controller's performance. For instance, the actual LiDAR sensor generates a point-cloud of data with a 240 degree field-of-view with a 0.36 degree resolution resulting in 682 beams or points of data. This is a significant amount of data. A much smaller number of LiDAR beams were simulated to increase the simulator's real-time performance and to maintain better update and rendering speeds. While experimenting with the number of points in the

LiDAR sensor's point-cloud, it was observed that a much smaller number of points could be used without detracting from the controller's ability to avoid collisions and to keep the vehicle within the roadway. This presents an opportunity to use cheaper LiDAR sensors with much lower resolution. This would also have the benefit of less data to process, which ultimately permits a higher closed-loop frequency. During experimental ground testing, the failure of the LiDAR sensor to accurately capture the surrounding environment was observed, which degraded the system performance. This occurred when in direct sunlight or exposed to heavy glare. This is an area for further research and investigation of some type of sensor redundancy or sensor fusion that may be required to cope with sensor failure or other disturbances. This section presented an example of how the simulation may be used to analyze the effect of the LiDAR's resolution on controller performance and proposed investigation of intermittent sensor failure on the controller's performance.

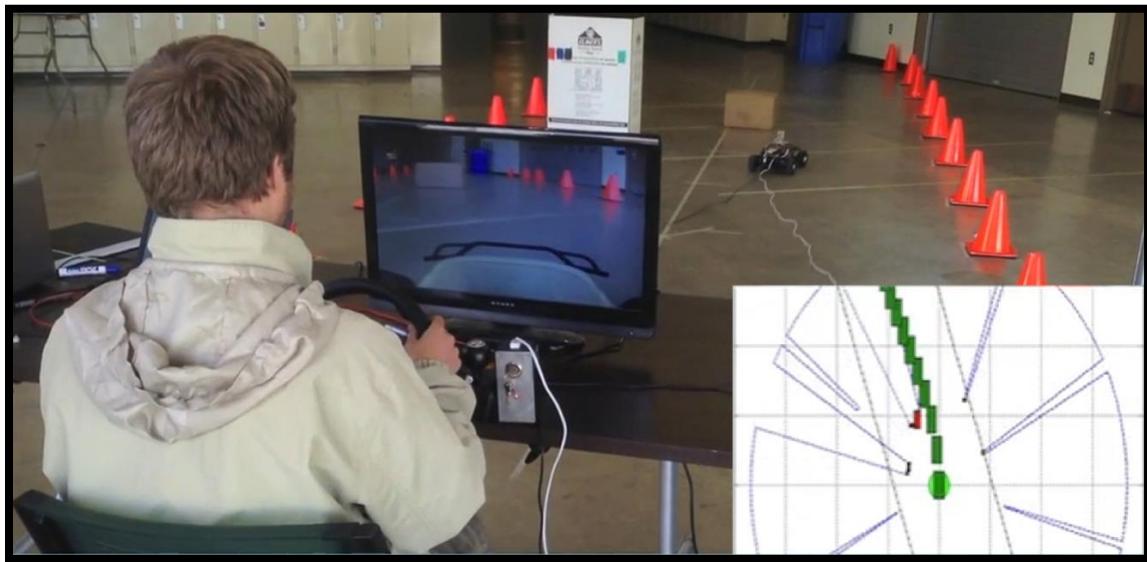


Figure 61. Previously implemented tethered system



Figure 62. Wireless system implementation

#### 5.4.3 Reducing Unjustified Controller Overrides By Improving Threat Determination

Most significantly, the simulation was used to alter the threat determination algorithm to realize better semi-autonomous operation of the system than was previously implemented in the tethered system. This included dramatically reducing the number of unjustified threatening events and the high switching frequency between user and computer control. In the previous wired implementation, a user was asked to drive down the roadway while attempting to strike an obstacle as shown in Figure 61. Once the system detected a threat and enabled the avoidance system, the avoidance maneuver was executed while maintaining a constant speed and then the vehicle was brought to a halt once the hazard was passed. In this configuration, the system never attempted to return control to the driver. This is because by the time the system enabled and executed an avoidance maneuver the roadway ended and the tether prevented further operation, which

prompted a need for the vehicle to stop. With a wireless configuration, shown in Figure 62, the controller was altered to allow the system to switch back to user control after an autonomous maneuver was performed. The influence of the detected road boundaries on the threat parameter was changed to allow more seamless and effective switching between driver and ECU control. During testing, it was determined that the detected road bounds would not allow the user to drive off the centerline of the roadway or near to the edge of the roadway without triggering a threat and controller override even if the vehicle was not in immediate danger of departing from the roadway. This threat was often unjustified and degraded the semi-autonomous aspect of the system's operation as well as the minimum interference principle of this control system. To address this concern, the detected roadway was divided into three threat areas: low, medium, and high. The threat region is determined from the position of the vehicle in the roadway. When the vehicle is in the low threat area, the distance to the road boundaries will never cause the system to trigger an override event and only hazards or impending collisions will cause the system to take control from the driver. If the car is in the medium threat area, the system will take over if the vehicle's detected yaw angle is determined to be too large in relation to the road bounds or if there is threat of an impending collision(s). Finally, if the vehicle is in the high threat area, the system will activate regardless of roadway hazards or impending collisions in order to stay within the road boundaries. This lane-keep assist functionality may also be disabled so that the system only reacts to detected obstacles. The threat regions are illustrated in Figure 63. An example of each threat region is shown in Figure 64.

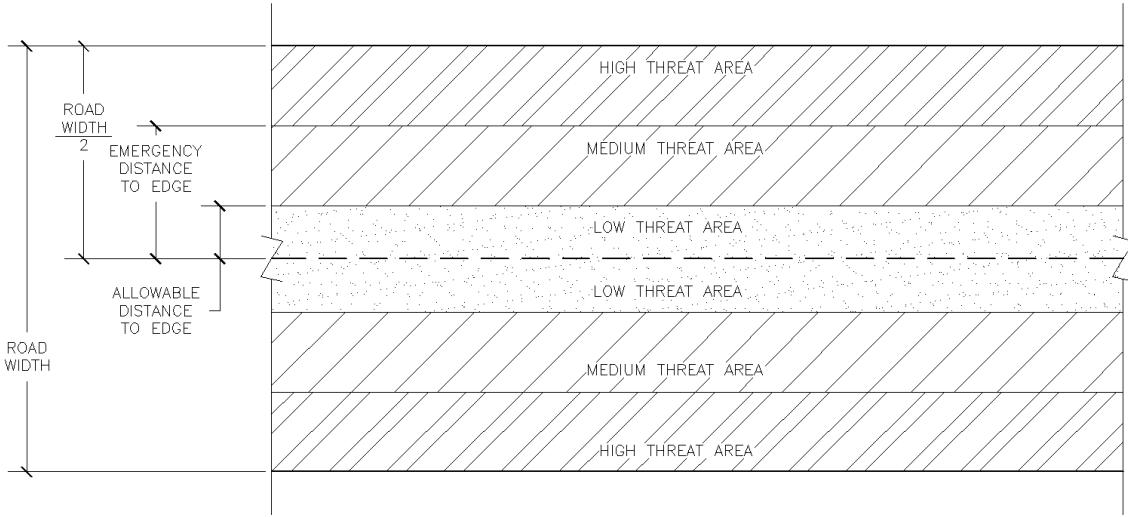


Figure 63. Threat determination diagram with respect to road boundaries

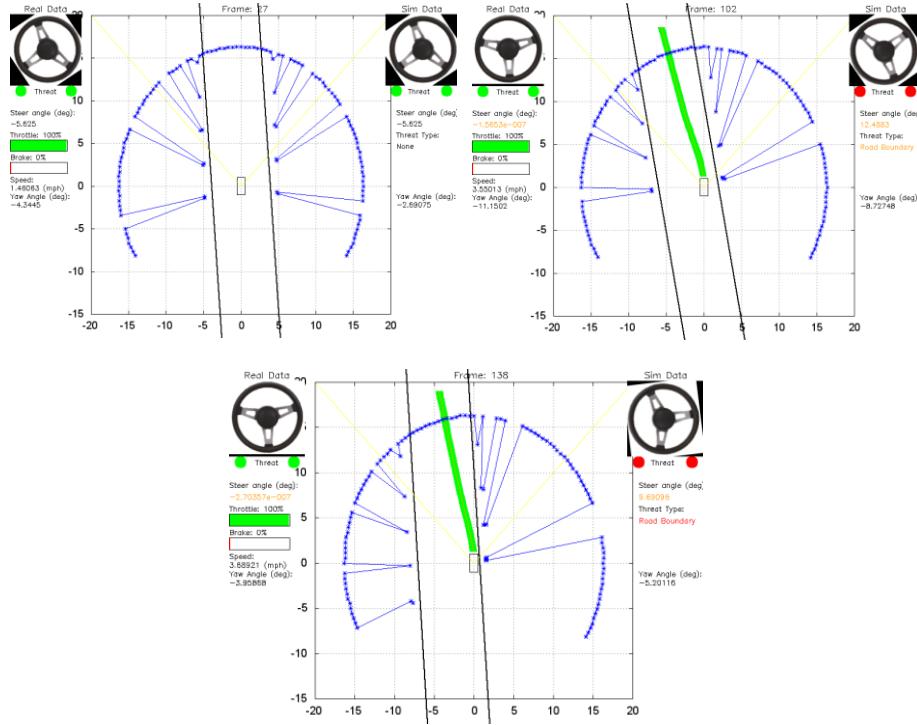


Figure 64. From left-to-right, low threat region example, medium threat region example and high threat region example

#### 5.4.4 Following the Same Path between Iterations of the RRT Algorithm

An attempt was made to improve the RRT path-planning algorithm to reduce the randomness of the algorithm's output that results in oscillatory steer commands,

especially when faced with a single obstacle in the center of the roadway. The oscillatory steer commands produce no net change in direction of the vehicle when avoiding an obstacle. This oscillatory behavior results from calculation of a new safe path every time the control-loop and RRT algorithm runs. This is especially apparent when the constraint data does not adequately force a general corridor in which to navigate through when planning a safe path. For instance, when an obstacle is in the center of the roadway the path-planning algorithm may plan a safe path left around the obstacle then subsequently plan a safe path right around the obstacle between control-loop iterations. This is shown in Figure 65. An effort to improve the RRT algorithm focused on reducing the resulting steer oscillations by only calculating a new safe path when the previously calculated safe path was deemed no longer safe to follow or the goal of the previous path was reached. While implementing this update, it was quickly realized that in order to follow the same path between controller iterations the change in vehicle position and heading must be considered and the previously calculated path must be compensated for by bringing it to the current frame of reference. This has to be considered as the LiDAR and obstacle data is seen in the current vehicle frame while the last calculated safe path is in the previous vehicle frame. The previous path must be checked for validity in the current frame for collisions with obstacles and to stay within the road boundaries. In order to bring the previous path to the current vehicle frame, the change in the vehicle's position and heading must be determined since the last iteration of the control-loop. Without the use of additional sensors, such as a GPS or IMU, the only way to estimate this is to use the available speed, elapsed time, and the detected yaw angle to dead-reckon the vehicle's position. Dead-reckoning is the process of determining the current position through

knowledge of the previous position and advancing it based upon the estimated speed over the elapsed time. This method is subject to cumulative error that increases with the distance from the original location. This method also assumes that the safe path is very closely followed. In addition, by using the same path between iterations and relying on dead-reckoning, the effectiveness of the closed-loop controller to correct for the oversimplified steer angle estimation to follow the path is greatly reduced. With additional sensors, a much more accurate and reliable way to determine the vehicle's position can be used to follow the previously calculated path if it is still safe to follow. In addition, a closed-loop controller that minimizes the error between the vehicle's actual path and the safe path could be implemented to better steer the vehicle. A vehicle tracking system could be simulated with the simulation tool described in chapter 4 to verify and develop these improvements before selecting, purchasing, and implementing any hardware components.

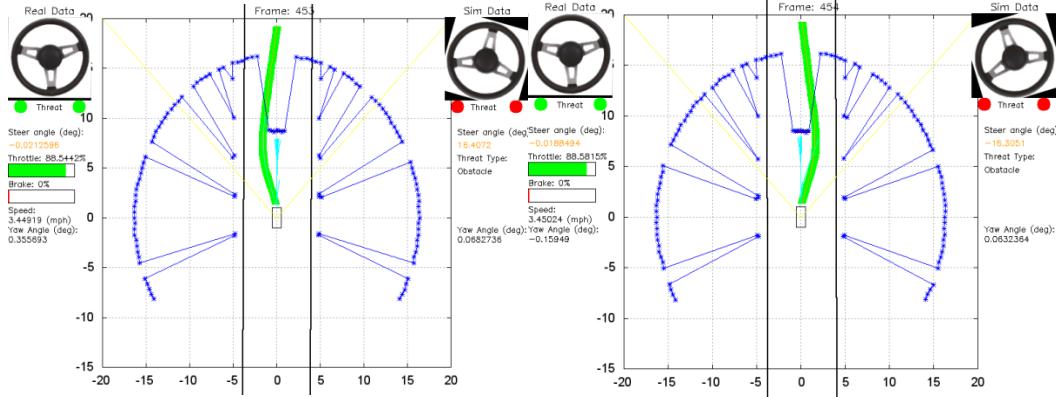


Figure 65. Subsequently planned paths around an obstacle showing cause of oscillatory steer commands

Another issue associated with following the same path is that the calculated path depends on the vehicle speed, which is assumed to be constant while planning the path. The path is stored as a lookup table with time values that indicate when the vehicle

should issue the next steer command in the table. With future updates, the time indexed steer lookup table may be compensated to allow for varying speeds while following the same path. This would allow the driver to control the throttle at all times during the autonomous maneuver while following the same path. The previous implementation of the system kept the throttle constant once a threat was detected and the system engaged. In addition, a completely new path is calculated for each iteration of the control-loop and only the first steer command in the look-up table is used. The wireless system also calculates a new path for every iteration of the control loop; since only the first steer command in the look-up table is used, no constant speed constraint or table compensation is required and the user remains in control of the vehicle's speed throughout the autonomous maneuver. This allows the driver to alter the speed at all times during system operation to better realize semi-autonomous operation; however, this feature can be disabled to mimic the previous tethered implementation.

With the current tuning of the threat determination, the system's behavior may be best described as a lane-keep assist system with steer-based collision avoidance. If the driver tries to go off of the roadway, the controller will attempt to fight the driver to bring the vehicle back to the center of the roadway; in the end, the driver can overpower the controller off of the roadway if the road boundary is approached with a speed and yaw angle. However, if the driver drives down the roadway with their hands completely off the steering wheel, the system easily maintains a central position in the roadway. This system behavior can be changed by tuning the distances to the road edge before controller performs a maneuver to stay on the roadway. Higher values will trigger a reaction from the controller sooner and a high enough value will essentially ignore all driver commands

in order to stay in the center of the roadway even if the driver pushes the steering wheel in an attempt to overpower the controller. If the threat to the vehicle is deemed imminent, then the controller takes over control and the haptic steering wheel effects merely provide feedback to the driver. In the event of a controller override the driver has no control over the vehicle even if they push the steering wheel with more force. The steering wheel angle may be used to implement a driver override feature if the driver's steering wheel angle greatly differs from that of the controller. One pitfall of this system is that if no road boundaries can be detected than the system cannot plan a safe path and therefore cannot avoid any collisions. A future test track may involve creation of a circular or oval track with evenly spaced road cones to prevent the road from ending. This scenario was simulated in the hardware in the loop simulator. As the road bounds are drawn as straight lines, some cones will appear in the center of the roadway as an obstacle while in a curve. This could be avoided by reducing the distance that an obstacle is considered threatening and curving the detected road lines in proportion to the yaw rate. The yaw rate may also be incorporated in determining the threat of a collision with particular object by extrapolating the vehicle's future path to see if the obstacle would be in its path. This could help provide a confidence metric within the threat determination algorithm with respect to each detected obstacle. The simulator can be used to investigate this further.

## 5.5 Embedded Controller Testing

Integral to this project and its wireless capability is the SBC that interfaces with all the sensor hardware, communicates with the user-control-station, and runs the collision avoidance algorithms. Implementing a real-time controller with an embedded Linux platform presents many challenges as it is not a real-time operating system. In

addition, the SBC has limited resources such as processing power and memory. The processing power and memory being used while running the open-loop and closed-loop controller consists of many tasks, as described in chapter 3. Instead of many distributed electronic control units (ECUs) running concurrently that share data on the vehicle's data bus, as is commonly found in modern automobiles, this system uses one ECU with many distributed FSMs running concurrently in separate threads that share data by utilizing a shared memory model. This is an advantage for scaled research as the footprint and power requirements are greatly reduced as well as the complexities of the hardware connections, integration, and synchronization of many different ECUs; although, this configuration does rely more heavily on the processor to perform all the tasks. Thus it is important that the SBC is adequately capable of performing all the required tasks without maxing out the processor or memory use as this could cause important real-time tasks to miss their deadlines and could degrade controller performance. The CPU load and memory use while operating the open-loop and closed-loop controller is shown in Figure 66. These figures show that the SBC is perfectly capable of running the controller's software without maxing out processor or incurring high memory use. A load average of 1.0 shows that the processor is fully utilized and processes do not have to wait to be scheduled. The load on the processor while running the open-loop controller is minimal; no task has to wait to be executed, as shown by the load average, and the processor is idle 95.6 percent of the time. For the closed-loop software, the average load on the processor shows that the software matches the processor's capability and is fully utilized without overloading it. The percentage of processing time is spent where anticipated, processing the point-cloud and determining the threat of a collision and a safe path to navigate. The

memory usage is also quite small and remains relatively constant for both the open and closed loop software, which helps show that the program does not contain any memory leaks. While the SBC is well suited to run the full system's software there is not as much room for further expansion of the software. For instance, additional features that use the same data could likely be incorporated without affecting the processor's ability to address the needs of each task; however, integration of new sensors, such as a camera for obstacle recognition and lane detection through image processing, would likely need additional hardware to avoid overloading the processor. There is also a significant opportunity to offload the load on the processor by utilizing the programmable real-time units and the enhanced quarture encoder package within the SoC. This and other possible improvements are discussed in the last chapter.



Figure 66. Controller's open and closed-loop CPU load and memory use

As Brennan states, the primary focus of using an automated controller to drive a vehicle during an emergency is to reduce the response time to an impending collision or instability. Often, the delay of a driver's response during an emergency situation is the biggest distinction between a poor and an expert driver. For an automated controller to be effective it must provide an appropriate response to avoid an impending collision, allow time to actuate the avoidance response, and allow enough time for the dynamic forces to develop. A semi-autonomous steering-based collision avoidance system's performance may be determined by the ability of the controller to safely steer a vehicle around an imminent threat in a variety of situations. In regards to the semi-autonomous aspect of the

controller, the switch between computer and user control has a dramatic influence on the driver's perception of this technology, especially if the switch is jarring and/or frightening to the driver. The switch must be fluent and seamless.

Determining the imminent threat to a vehicle is a complicated matter, especially when this threat determination dictates whether the driver or the vehicle controller is in control of the vehicle. There are many unknowns that could influence the threat to a vehicle involving the environment, road surfaces, other vehicles, and other driver's behavior that can be extremely difficult to model and predict. To compound the complications of quantifying a threat level, the vehicle's hardware has limited processing power, sensor range, and must perform within strict time constraints. Properly identifying a threat is imperative as failure to do so within time may put the driver, by-standers, and the vehicle's hardware at great risk. Failure to determine a valid threat may result in injury or death. Conversely, falsely identifying a threat may result in oscillating between driver and computer control, which could be jarring to the driver and could be just as dangerous as missing a valid threat altogether. There are many obstacles that a vehicle may be presented with, such as: moving & stationary vehicles, people, animals, roadway barriers, and road surface conditions. The focus of this project was narrowed by only differentiating road boundaries from other obstacles. A numerical quantification of each obstacle's threat metric was compared with a threshold threat metric to determine if this obstacle should be added to the threatening obstacle set for consideration during path-planning. Future development could account for varying obstacle types, pedestrians, and their behaviors.

This chapter showed the results from the hardware in the loop simulator and the implementation of a Linux based embedded controller on the vehicle platform. The wireless implementation of the controller greatly expanded the vehicle's capacity to study and develop collision avoidance software and to research human factors. The SBC implemented as the vehicle controller that runs the collision avoidance software in real-time in a multithreaded FSM configuration was shown to have adequate resources. This showed that an SBC running Linux is a viable option for a semi-autonomous collision avoidance system. This chapter presented some the usefulness of hardware in the loop simulation for identifying issues in the collision avoidance software as well as further developing the underlying algorithms. Specifically, the issue of switching between driver and computer control through the determination of a threat metric was investigated and refined. The reasons behind path-planning algorithms oscillatory behavior were also examined. In the next chapter, recommendations are suggested to improve system performance by expanding the functionality of the vehicle's controller and its algorithms.

## **Chapter 6 Conclusion and Recommendations**

Active safety and autonomous collisions avoidance systems are very complex to say the least. With dynamic, hardware, computing, and sensory considerations these systems cover a broad range of engineering disciplines and exhibit non-linear behaviors that make them difficult to design and evaluate. Interesting results often only appear after complete immersion and hours of testing. The primary purpose of this project's crash avoidance system is to serve as a research platform to ascertain how drivers interact and perceive this technology and to further develop the collision avoidance algorithms to better coincide with driver's expectations. It is important that driver's perception of this technology does not hinge on the specific implementation or algorithms but rather on the technology as a whole. The simulator and analysis toolchain created for this project help to facilitate development of the system to help ensure that this is the case. Active safety systems and vehicle automation has the potential to save thousands of lives and billions of dollars. The degree to which drivers are ready to embrace this technology in their own vehicles is a significant factor in achieving widespread implementation and hinges on how well the controller performs.

Through testing it was seen that the threat determination, and thus the switch between driver and computer control, plays a crucial role in determining the performance and behavior of a semi-autonomous collision avoidance system. For example, a very aggressively tuned threat leads the vehicle to exude more fully autonomous behavior where the driver's steer inputs are largely ignored. A lightly tuned threat yields a system with functionality that more closely resembles driver assistance behavior where the controller's effect may only be realized by the driver if they completely take their hands

off of the wheel. A lightly tuned threat keeps the vehicle in the center of the roadway; however, the driver inputs can win out forcing the vehicle to exit the roadway. Both of these threat behaviors can be traced back to the binary switch between computer and driver control and the frequency at which this switch occurs. The more constrained the threat is, the more fully autonomous behavior is observed with less frequent switching of control back-and-forth between the driver and the computer and vice-versa. Hence, there is a balance with respect to the sensitivity of the threat when tuning the controller, which depends on the desired performance. As described in (Anderson, Peters, Pilutti, Tseng, & Iagnemma), alternative controller schemes exist that simultaneously consider the computer and driver steer inputs. Instead of a binary switch between computer and driver control, a proportion of control is given to both by weighting the computer and driver control commands depending on the threat level. For each iteration of the control-loop, the wheel sideslip is converted into a scalar threat metric that is used in a piecewise-linear intervention function to blend the driver and controller inputs as shown in Equation 36.

$$\mathbf{u}_{\text{vehicle}} = \mathbf{K}(\emptyset)\mathbf{u}_{\text{Controller}} + (1 - \mathbf{K}(\emptyset))\mathbf{u}_{\text{driver}} \quad (36)$$

As implemented, this steering-based collision avoidance system does not consider the throttle as a control parameter. This greatly complicates the use of the calculated steering commands necessary to follow the same path planned by the RRT algorithm between control-loop iterations. These steering commands are calculated based on the current speed of the vehicle. If the vehicle's speed changes the required steering command to follow the same path must also change, as is reflected in the pure-pursuit control law discussed in chapter 2. For now, only the first steer command is used since it is calculated using the current vehicle speed. A new safe path is calculated every time the

control-loop runs regardless of whether the previous path is still valid. While this allows the driver to be in control of the throttle and speed of the vehicle at all times, it is inefficient in terms of processing. Furthermore, if the throttle and speed is not held constant while performing an autonomous maneuver the dynamic region in which the controller operates may change. This can greatly affect the necessary controller output and the vehicle's dynamic response. A controller used to avoid imminent collisions with static and moving obstacles would likely be the most effective if all control inputs were considered including braking, throttle, and steering. While this consideration is out of the scope of this project, there is great potential in using all control signals to optimize the avoidance maneuver for safety and comfort. From analysis of the paths generated by the RRT path-planning algorithm, it is apparent that it is not an optimal path-planning algorithm. The RRT can only perform so many iterations in a given amount of time and does not perform consistently with the same constraint data due to the random nature of the algorithm. Several iterations may look in the same area to place the next node despite having already failed to place a node in that area. In addition, the driver's input has no influence on the generated path, which may be jarring to the driver if they maneuver one way to avoid an obstacle and the controller overrides the driver with a maneuver in the other direction to avoid the same obstacle. The hardware-in-the-loop simulator could be used to conduct a study to improve components of the RRT algorithm as well as investigate the viability of alternative path-planning algorithms. Updating the RRT path-planning algorithm to follow the same path or biasing the generated path to consider the user's input and previously searched regions could dramatically improve the controller's ability to avoid collisions. Alternatively, implementation of an efficient and optimal path-

planning algorithm, such as artificial potential fields (APF) seen in Figure 67, may provide more consistent results and improve performance. As discussed in (Michielsen, 2013), APF methods have been studied extensively and are extremely powerful and applicable for autonomous vehicle path-planning applications. With little computational overhead, APF path-planning algorithms have great potential for use in real-time semi-autonomous collision avoidance systems.

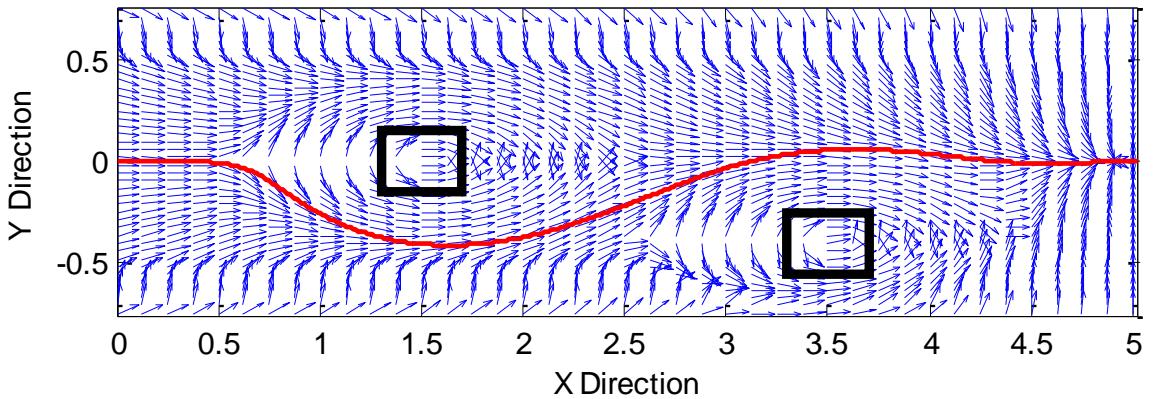


Figure 67. Path-planning with artificial potential fields

As discussed in chapter 3, an interrupt driven approach was taken in order to measure the speed of the vehicle with the wheel encoders. While this method is adequate at low speeds, it becomes increasing difficult to keep an accurate count of the wheel encoder ticks as the wheel speeds increase. With interrupts, a portion of the processor's CPU cycles must be used to monitor the wheel encoder interrupts and count the encoder ticks, which detracts from the main program running the collision avoidance software. An alternative and more efficient approach could be taken by utilizing the three enhanced quadrature encoder package (eQEP) hardware blocks within the SoC. This would help reduce the load on the main processor while enabling higher wheel encoder resolutions without sacrificing CPU cycles. Unfortunately, this does not circumvent the use of

interrupts altogether as there are only a limited number of these modules within the AM335x processor; only three wheel encoders may be monitored with this method. An even better solution would be to use one of the programmable real-time units (PRU) to completely void the main processor of monitoring the wheel encoders. The BeagleBone has two PRUs which are completely separate 200MHz cores within the processor that offer hard real-time determinism and have the great potential for further development. These cores both have their own memory for data and instructions and also share memory with the main processor to facilitate communication. The PRUs may be used to incorporate other sensors with a high degree of resolution or to handle the serial communication with the user control station. While the PRUs are programmed in assembly, an optimized C/C++ compiler was recently released by Texas Instruments to generate the assembly code.

This system uses the LiDAR sensor to obtain point-clouds of data that represent the surrounding environment. These point-clouds are segmented and obstacles are parsed out. The obstacle properties are searched in order to differentiate between hazardous obstacles and cone obstacles that mark the road boundaries. The CMOS camera is merely used to feed a visual driver perspective to the user control station's monitor. Future development of this system could incorporate the camera for use in image processing based algorithms to enhance system performance and functionality. With image processing, the detected obstacles could be classified and the road edges could be detected with lane markings. This could also help to track the different obstacles and to better estimate their speeds and headings. A recent FPGA cape for the BeagleBone could be used to facilitate the incorporation of a camera and to run the image processing

algorithms. Another interesting cape designed for the BeagleBone, called the Interacto, uses one of the PRUs to capture images from a small camera at thirty frames per second and stores the results in data shared with the main processor. This allows hard real-time image capture without any load on the main CPU. Needless to say, the BeagleBone is a very capable SBC with great opportunity for further expansion.

As mentioned above, investigation of the system's performance around a curved roadway or oval track would be an interesting study. The vehicle's yaw rate could be measured and used to estimate the curvature of the detected roadway. The hardware-in-the-loop simulator could be used to develop and simulate these features and also has great potential for expansion. For instance, multiple controllers could be used in the simulation while simulating multiple vehicles with the same system to observe how fleets of vehicles with the same technology would behave. With respect to the vehicle hardware, there is significant capacity to improve the scaled vehicle's dynamic similarity to a full size vehicle. By reconfiguring the hardware, adding weights in appropriate locations, and changing the tires on the vehicle better similarity could be achieved by matching the pi groups between a full size vehicle and the scaled vehicle platform for better dynamic similarity, as described by Brennan.

While moving obstacles are not explicitly considered in the current implementation, this is still an effective system with the potential to greatly reduce fatalities and injuries resulting from distracted drivers. Nothing prevents this system from avoiding moving obstacles; however, the path-planning algorithm does not consider the velocity of each obstacle as the segmentation algorithm does not adequately estimate each obstacle's velocity. The effectiveness of this algorithm to avoid moving obstacle is

degraded by the reduction in time available to react and no consideration of the obstacle's speed relative to the vehicle. Even without consideration of moving obstacles, avoiding collision with static obstacles is an effective active safety technology. To put this into perspective, in 1988 44.73% of fatalities occurred from vehicles colliding with inanimate, stationary objects not found on the road (Brennan, 1997).

Automation plays a massive role in society as airplanes, vehicles, manufacturing processes, and repetitive tasks are being continuously replaced with electronic control systems that offer increases in efficiency, throughput, and safety. The way in which people interact and perceive automated systems is greatly important, especially in regards to the automation of vehicles and their navigation. Car companies are continually improving vehicle's safety by releasing innovative products that sense, react, and automate the operation of a vehicle. One of the most complicated aspects of active vehicle safety systems, especially semi-autonomous systems, is blending them with humans. If these systems are implemented in such ways that the driver sees them as irritating or as not controlling the vehicle as a human would they may disable these systems altogether or opt for vehicles that do not have them equipped. As described in (Richtel & Dougherty, 2015), an example of this is lane departure warning systems where some drivers found the acoustic beeping as being irritating when they purposefully changed lanes without a turn signal. The blending of robot and human drivers also presents some interesting challenges. For example, drivers tend to look at each other at intersections, pedestrians tend to look to see if drivers are paying attention to them before crossing the street, and robot cars don't possess eyes. Despite these hurdles, active safety systems are already available in vehicles and fully automated vehicles are soon to follow.

Active safety systems will likely become increasingly prevalent as the progress to fully autonomous vehicles continues. This is certainly a very exciting and active area of research that is already helping to save lives with the potential to save many more. Vehicle collisions, especially those that cause severe injuries and fatalities, will soon go the way of the dinosaur.

## BIBLIOGRAPHY

- Ahmad, S. (2013, May 31). *Using PWM on the BeagleBone Black*. Retrieved from <http://www.saadahmad.ca/using-pwm-on-the-beaglebone-black/>
- Anderson, S. J., Peters, S. C., Pilutti, T. E., Tseng, H., & Iagnemma, K. (n.d.). *Semi-Autonomous Avoidance of Moving Hazards for Passenger Vehicles*.
- Brennan, S. N. (1997). *Modeling and Control Issues Associated with Scaled Vehicles*. Urbana, Illinois.
- CarSim. (n.d.). CarSim Mechanical Simulation. Retrieved from <https://www.carsim.com/products/carsim/>
- Columbia University. (n.d.). Linux Scheduler. Retrieved from <https://www.cs.columbia.edu/~smb/classes/s06-4118/113.pdf>
- Daimler. (n.d.). PRE-SAFE Brake: The Anticipatory Brake. Retrieved from <http://www.daimler.com/dccom/0-5-1210220-1-1210348-1-0-0-1210338-0-0-135-0-0-0-0-0-0-0.html>
- Dash, N. P., Dasgupta, R., Chepada, J., & Halder, A. (2011). *Event Driven Programming for Embedded Systems - A Finite State Machine Based Approach*. Kolkata, India: Innovation Lab.
- Davies, A. (2014, June). Infiniti's New Steering System Is a Big Step Forward - Unless You Love Cars. Retrieved from <http://www.wired.com/2014/06/infiniti-q50-steer-by-wire/>
- Dixon, J. C. (1996). *Tires, Suspension, and Handling* (2nd ed.).
- Free Electrons. (2015, January). Linux Kernel and Driver Development Training. Retrieved from <http://free-electrons.com/doc/training/linux-kernel>
- Free Electrons. (n.d.). *Embedded Linux system development*. Retrieved from Free Electrons Web site: <http://free-electrons.com/doc/training/embedded-linux>
- Gillespie, T. D. (n.d.). *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, Inc.
- Lipari, G. (2009, May 12). *Programming RT systems with pthreads*. Retrieved from <http://www.feanor.sssup.it/~lipari>
- Michielsen, J. (2013). *Crash Avoidance Path Planning using Artificial Potential Fields*. M.S. Thesis, California Polytechnic State Univ.

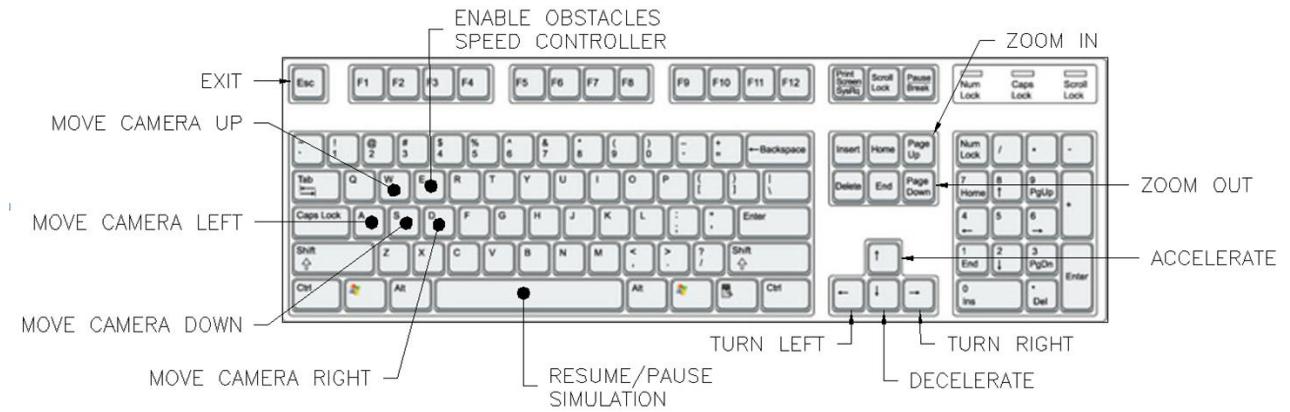
- Milgram, J. (2003). Numerical Integration of Differential Equations. Retrieved from  
[http://www.myoops.org/cocw/mit/NR/rdonlyres/Ocean-Engineering/13-024Spring2003/09A5CF66-2BA1-48D4-92E6-70EF1F26A809/0/differential\\_eqs.pdf](http://www.myoops.org/cocw/mit/NR/rdonlyres/Ocean-Engineering/13-024Spring2003/09A5CF66-2BA1-48D4-92E6-70EF1F26A809/0/differential_eqs.pdf)  
[http://www.myoops.org/cocw/mit/NR/rdonlyres/Ocean-Engineering/13-024Spring2003/09A5CF66-2BA1-48D4-92E6-70EF1F26A809/0/differential\\_eqs.pdf](http://www.myoops.org/cocw/mit/NR/rdonlyres/Ocean-Engineering/13-024Spring2003/09A5CF66-2BA1-48D4-92E6-70EF1F26A809/0/differential_eqs.pdf)
- Miller, J., Ujiie, B., & Woods, G. (2011). *Autonomous Crash Avoidance System*. Senior Project Report, California Polytechnic State University, San Luis Obispo.
- Milliken, W. F., & Milliken, D. L. (1995). *Race Car Vehicle Dynamics*. Society of Automotive Engineers.
- Monster, M. (2003, November). Car Physics for Games. Retrieved from  
<http://www.asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html>
- MSC Software. (n.d.). Adams/Car: Real Dynamics for Vehicle Design and Testing. Retrieved from [http://www.mscsoftware.com/sites/default/files/ds\\_adams-car\\_ltr\\_w\\_0.pdf](http://www.mscsoftware.com/sites/default/files/ds_adams-car_ltr_w_0.pdf)
- National Highway Traffic Safety Administration. (2000). *Economic Impact of Motor Vehicle Crashes 2000*. U.S. Department of Transportation, Washington, D.C.
- National Highway Traffic Safety Administration. (2007). *Federal Motor Vehicle Safety Standards; Electronic Stability Control Systems; Controls and Displays - Final Ruling*. U.S. Department of Transportation, Washington, D.C.
- National Highway Traffic Safety Administration. (2009). *Fatalities in Frontal Crashes Despite Seat Belts and Air Bags*. Washington, D.C.: U.S. Department of Transportation.
- National Highway Traffic Safety Administration. (2010). *Traffic Safety Facts 2010*. U.S. Department of Transportation, Washington, D.C.
- Nise, N. S. (2011). *Control Systems Engineering* (6 ed.). John Wiley & Sons, Inc.
- Noxon, N. (2012). *A Model Predictive Control Approach to Roll Stability of a Scaled Crash Avoidance Vehicle*. M.S. Thesis, California Polytechnic State Univ., Dept. Mech. Eng., San Luis Obispo.
- Ohio University. (2008, October). Ohio University driving simulation lab to improve traffic, construction zone safety. Retrieved from  
[http://www.ohio.edu/research/communications/driving\\_simulation.cfm](http://www.ohio.edu/research/communications/driving_simulation.cfm)

- Oracle. (2014). The Java Tutorials: Double Buffering and Page Flipping. Retrieved from <http://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html>
- Pacejka, H. B. (2002). *Tire and Vehicle Dynamics*.
- Palm |||, W. J. (2007). *Mechanical Vibration*. Hoboken, NJ: John Wiley & Sons, Inc.
- Palopoli, L. (n.d.). Real-Time Embedded Control. Retrieved from [http://www.disi.unitn.it/~palopoli/hycon2\\_slides.pdf](http://www.disi.unitn.it/~palopoli/hycon2_slides.pdf)
- Richtel, M., & Dougherty, C. (2015, September 1). Google's Driverless Cars Run Into Problem; Cars With Drivers. *The NY Times*.
- Snider, J. M. (2009). *Automatic Steering Methods for Autonomous Automobile Path Tracking*. Carnegie Mellon University, Pittsburgh.
- Sparkfun. (n.d.). *Bluetooth Basics*. Retrieved from Sparkfun: [https://learn.sparkfun.com/tutorials/bluetooth-basics?\\_ga=1.228314781.1047042560.1418342781](https://learn.sparkfun.com/tutorials/bluetooth-basics?_ga=1.228314781.1047042560.1418342781)
- Stan, A., Botezatu, N., Panduru, L., & Lupu, R. G. (2009). *A Finite State Machine Model Used in Embedded Systems Software Development*. Universitatea Tehnica, Gheorghe Asachi din Iasi.
- Steele, J. (1955). Motion Sickness. *Shock and Vibration Bulletin*(22).
- Stevens, T., Birdsong, D., Painter, I., & Carlson, E. (2013). *Overview of a Student Semi-Autonomous 1/10th Scale Vehicle Collision Avoidance Platform*. California Polytechnic State University San Luis Obispo, San Luis Obispo.
- Stevens, T., Carlson, E., & Painter, I. (2013). *Autonomous Collision Avoidance Final Design Report*. Senior Project Report, California Polytechnic State University, San Luis Obispo.
- Tass International. (n.d.). PreScan Overview. Retrieved from <https://www.tassinternational.com/prescan-overview>
- Taylor, P., Funk, C., & Craighill, P. (n.d.). *Americans and Their Cars: Is the Romance on the Skids?* A Social Trend Report, Pew Research Center.
- Texas Instruments. (2011). *AM335x ARM Cortex-A8 Microprocessors (MPUs): Technical Reference Manual*.
- The Cooper Firm. (2014, March). How Can Electronic Stability Control Save Your Life? Retrieved from <http://thecooperfirm.com/can-electronic-stability-control-esc-save-life/>

- Vahid, F., & Givargis, T. (1999). *Embedded System Design: A Unified Hardware/Software Approach*. University of California Riverside, Riverside.
- Weilkes, M., Burkle, L., Rentschler, T., & Scherl, M. (2005, January). Future vehicle guidance assistance - combined longitudinal and lateral control. *Automatisierungstechnik*, 42, 4-10.
- Wymann, B. (n.d.). About TORCS. Retrieved from  
<http://torcs.sourceforge.net/index.php?name=Sections&op=viewarticle&artid=1>
- Yang, J. (n.d.). W4118 Operating Systems. Retrieved from  
<http://www.cs.columbia.edu/~junfeng/10sp-w4118/lectures/l14-sched-linux.pdf>

## APPENDICES

### Appendix A Hardware-in-the-Loop Simulation Controls



Simulation Keyboard Controls

## Appendix B Embedded System Code

```
boot/uboot/uEnv.txt

##Video: Uncomment to override:
##see:
https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/fb/modedb.txt
#kms_force_mode=video=HDMI-A-1:1024x768@60e

##Enable systemd
systemd=quiet init=/lib/systemd/systemd

##BeagleBone Cape Overrides

##BeagleBone Black:
##Disable HDMI/eMMC
#cape_disable=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN,BB-BONE-EMMC-2G

##Disable HDMI
cape_disable=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN

##Audio Cape (needs HDMI Audio disabled)
#cape_disable=capemgr.disable_partno=BB-BONELT-HDMI
#cape_enable=capemgr.enable_partno=BB-BONE-AUDI-02

##AVOIDANCE CAPE
cape_enable=capemgr.enable_partno=BB-UART4

##Example
#cape_disable=capemgr.disable_partno=
#cape_enable=capemgr.enable_partno=

##WIP: v3.14+ capes..
#cape=ttyO1
#cape=

##note: the eMMC flasher script relies on the next line
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait fixrtc

##These are needed to be compliant with Angstrom's 2013.06.20 u-boot.
console=ttyO0,115200n8

kernel_file=zImage
initrd_file=initrd.img
```

```

loadaddr=0x82000000
initrd_addr=0x88080000
fdtaddr=0x88000000

initrd_high=0xffffffff
fdt_high=0xffffffff

loadkernel=load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${kernel_file}
loadinitrd=load mmc ${mmcdev}:${mmcpart} ${initrd_addr} ${initrd_file}; setenv
initrd_size ${filesize}
loadfdt=load mmc ${mmcdev}:${mmcpart} ${fdtaddr} /dtbs/${fdtfile}

loadfiles=run loadkernel; run loadinitrd; run loadfdt
mmcargs=setenv bootargs console=tty0 console=${console} ${optargs} ${cape_disable}
${cape_enable} ${kms_force_mode} root=${mmcroot} rootfstype=${mmcrootfstype}
${systemd}

uenvcmd=run loadfiles; run mmcargs; bootz ${loadaddr} ${initrd_addr}:${initrd_size}
${fdtaddr}
#
# etc/profile

# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ `id -u` -eq 0 ]; then
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else

    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/ga
mes"
fi
export PATH

if [ "$PS1" ]; then
    if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
        # The file bash.bashrc already sets the default PS1.
        # PS1='h:\w\$ '
        if [ -f /etc/bash.bashrc ]; then
            . /etc/bash.bashrc
        fi
    else
        if [ `id -u` -eq 0 ]; then
            PS1='# '

```

```

else
    PS1='\$ '
fi
fi
fi

# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.

if [ -d /etc/profile.d ]; then
    for i in /etc/profile.d/*.*; do
        if [ -r $i ]; then
            . $i
        fi
    done
    unset i
fi

export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
export SLOTS=/sys/devices/bone_capemgr.9/slots

etc/resolv.conf

domain localdomain
search localdomain
nameserver 192.168.1.1
nameserver 8.8.8.8

etc/default/capemgr
# Default settings for capemgr. This file is sourced by /bin/sh from
# /etc/init.d/capemgr.sh

# Options to pass to capemgr
CAPE=bspm_P9_42_2f,AVOIDANCE_GPIO

lib/system/system/SoftwareSelectorService.service

[Unit]
Description=Service to select whether to start RRT avoidance software or just bluetooth
driver software if button is not pressed
After=network.target

[Service]
Type=simple
ExecStart=/root/SoftwareSelectorService.sh
Restart=always

```

```
RestartSec=1
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
root/AVOIDANCE_GPIO-00A0.dts
```

```
/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Purpose License Version 2 as
 * published by the Free Software Foundation
 *
 * maps GPIO pins
 */

/dts-v1/;
/plugin/;

/{
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part-number = "AVOIDANCE-GPIO";
    version = "00A0";

    fragment@0 {
        target = <&am33xx_pinmux>;

        __overlay__ {
            pinctrl_test: AVOIDANCE_GPIO_Pins {
                pinctrl-single,pins = <

                    /*0x078 0x07*/ /* P9_12 60 OUTPUT MODE7 - The
LED Output */
                    /*0x184 0x2f */ /* P9_24 15 INPUT MODE7 none - The Button Input */
                    /*0x034 0x37 */ /* P8_11 45 INPUT MODE7 pullup - Yellow Wire */
                    /*0x030 0x27 */ /* P8_12 44 INPUT MODE7 pulldown - Green Wire */
                    /*0x024 0x2f */ /* P8_13 23 INPUT MODE7 none - White Wire */

                0xa4 0x37
                0xac 0x37
            };
        };
    };
}
```

```

/* OUTPUT GPIO(mode7) 0x07 pulldown, 0x17 pullup, 0x?f no
pullup/down */
/* INPUT GPIO(mode7) 0x27 pulldown, 0x37 pullup, 0x?f
no pullup/down */

        >;
    };
};

fragment@1 {
    target = <&ocp>;
    __overlay__ {
        test_helper: helper {
            compatible = "bone-pinmux-helper";
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_test>;
            status = "okay";
        };
    };
};

```

```
bspm_P9_42_2f_00A0.dts

/*
 * This is a template-generated file from BoneScript
 */

/dts-v1/;
/plugin/;

/{
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part_number = "BS_PINMODE_P9_42_0x2f";

    exclusive-use =
        "P9.42",
        "gpio0_7";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            bs_pinmode_P9_42_0x2f: pinmux_bs_pinmode_P9_42_0x2f {
                pinctrl-single,pins = <0x164 0x2f>;
            };
        };
    };
}
```

```

    };
};

fragment@1 {
    target = <&ocp>;
    __overlay__ {
        bs_pinmode_P9_42_0x2f_pinmux {
            compatible = "bone-pinmux-helper";
            status = "okay";
            pinctrl-names = "default";
            pinctrl-0 = <&bs_pinmode_P9_42_0x2f>;
        };
    };
};

```

root/BluetoothDriverService.sh

```

#!/bin/bash
/root/src/RealTime/BluetoothDriverwithSpeed_RT -D /dev/ttyO4

```

root/FullAutonomousService.sh

```

#!/bin/bash
/root/src/RealTime/FullSys_RT -D /dev/ttyO4

```

root/SoftwareSelectorService.sh

```

#!/bin/bash

```

```

# echo "AVOIDANCE_GPIO" > "/sys/devices/bone_capemgr.9/slots"

```

```

path1="/sys/class/gpio/gpio71/value"

```

```

LED1="/sys/class/gpio/gpio77/value"
LED2="/sys/class/gpio/gpio75/value"
LED3="/sys/class/gpio/gpio80/value"
LED4="/sys/class/gpio/gpio81/value"

```

```

if [ ! -f $path1 ]
then
    echo 71 > "/sys/class/gpio/export"
fi

```

```

value=$(cat $path1)

```

```

if [ ! -f $LED1 ]
then
echo 77 > "/sys/class/gpio/export"
echo "out" > "/sys/class/gpio/gpio77/direction"
fi

if [ ! -f $LED2 ]
then
echo 75 > "/sys/class/gpio/export"
echo "out" > "/sys/class/gpio/gpio75/direction"
fi

if [ ! -f $LED3 ]
then
echo 80 > "/sys/class/gpio/export"
echo "out" > "/sys/class/gpio/gpio80/direction"
fi

if [ ! -f $LED4 ]
then
echo 81 > "/sys/class/gpio/export"
echo "out" > "/sys/class/gpio/gpio81/direction"
fi

if [ -f $path1 ]
then
if [ "$value" -lt 1 ]
then
# echo BB-UART1 > "/sys/devices/bone_capemgr.9/slots"
echo "Button pressed - launch Full System Collision Avoidance Software"
for i in `seq 1 5`;
do
echo 1 > $LED1
sleep 1
echo 0 > $LED1
sleep 1
done
echo 0 > $LED2
echo 1 > $LED1
sh /root/FullAutonomousService.sh
else
# echo BB-UART1 > "/sys/devices/bone_capemgr.9/slots"
echo "Button not pressed - launch BluetoothDriver Software"
for i in `seq 1 5`;
do
echo 1 > $LED2

```

```

sleep 1
echo 0 > $LED2
sleep 1
done
echo 0 > $LED1
echo 1 > $LED2
sh /root/BluetoothDriverService.sh
fi
fi

root/src/BeagleBone/include/RealTime/Tasks/globaldata.h

#ifndef GLOBAL_DATA_H_
#define GLOBAL_DATA_H_

#include "encoder.h"
#include "PWM.h"
#include <termios.h>
#include <queue>
#include <iostream>

// #define DEBUG
#define STEERPIN "P8_13"
#define THROTTLEPIN "P8_19"

extern volatile bool globalRUN; // declared in file where main() is
located
extern pthread_mutex_t global_RUN_mutex; // declared in file where main() is
located
extern char* modemdevice; // declared in
monitorbluetooth.cpp - set from main program input args
extern speed_t baudrate; // declared in
monitorbluetooth.cpp - set from main program input args
extern double poll_timeout; // declared in monitorbluetooth.cpp -
set from main program input args

extern volatile float avgsped; // declared in monitorecoders.cpp
extern pthread_mutex_t encoders_mutex; // declared in monitorecoders.cpp

extern volatile int throttleInt; // declared in motorupdater.cpp
extern volatile int steerInt; // declared in motorupdater.cpp
extern pthread_mutex_t motor_update_mutex; // declared in motorupdater.cpp
extern pthread_cond_t motor_update_cond; // declared in motorupdater.cpp

extern volatile int threat;
extern pthread_mutex_t threat_mutex;

```

```

extern volatile int validPath;

extern int fullAuto;
extern int holdThrottle;
extern int indicateFreq;
extern int logOption;
extern std::queue<std::string> logdataqueue;
extern pthread_mutex_t log_data_mutex;

#endif

```

root/src/BeagleBone/include/RealTime/Tasks/constants.h

```

#include <math.h>

const unsigned int FL_ENCODER_PIN = 8;//46;
const unsigned int FR_ENCODER_PIN = 9;//65;
const unsigned int RL_ENCODER_PIN = 44;//26;
const unsigned int RR_ENCODER_PIN = 7;
const float ENCODER_TICKS = 32.0;

const float STEER_NEUTRAL = 0.14f;
const float THROTTLE_NEUTRAL = 0.15f;

const int LED1 = 71;
const int LED2 = 75;
const int LED3 = 80;
const int LED4 = 81;

const float wheelRadius = 0.1667f; // wheel radius in feet
const float wheelCircumference = ( 2.0 * M_PI * wheelRadius );
const float encoderStepsize = ( wheelCircumference / ENCODER_TICKS );

const float wheelBaseWidth = 1.0; // wheel base width in feet

```

root/src/BeagleBone/include/RealTime/Tasks/time-math.h

```

#ifndef TIME_MATH_UTILS_H_
#define TIME_MATH_UTILS_H_

inline void timespec_add_us(struct timespec *t, long us) {
    t->tv_nsec += us*1000;
    if (t->tv_nsec > 1000000000) {
        t->tv_nsec = t->tv_nsec - 1000000000;// + ms*1000000;
        t->tv_sec += 1;
    }
}

```

```

    }
    inline int timespec_cmp(struct timespec *a, struct timespec *b) {
        if (a->tv_sec > b->tv_sec) return 1;
        else if (a->tv_sec < b->tv_sec) return -1;
        else if (a->tv_sec == b->tv_sec) {
            if (a->tv_nsec > b->tv_nsec) return 1;
            else if (a->tv_nsec == b->tv_nsec) return 0;
            else return -1;
        }
    }
}

#endif

```

root/src/BeagleBone/include/RealTime/Tasks/PWM.h

```

/*
 * PWM.h
 * A C++ wrapper for the EHRPWM interface
 *
 * Created on: May 27, 2013
 *      Author: Saad Ahmad ( http://www.saadahmad.ca )
 */

```

```

#ifndef PWM_H_
#define PWM_H_

#include <string>
#include <dirent.h>
#include <fstream>
#include <sstream>
#include <iostream>

const int MICRSECONDS_TO_NANOSECONDS = 1000;
const int MILLISECONDS_TO_MICROSECONDS = 1000;
const int MILLISECONDS_TO_NANOSECONDS =
    MILLISECONDS_TO_MICROSECONDS * MICRSECONDS_TO_NANOSECONDS;
const long MODULE_DELAY_TIME_US = 100 *
    MILLISECONDS_TO_MICROSECONDS; // Time to wait for module to be loaded and
    // the sysfs interface setup

#define DEBUG_VERBOSE_OUTPUT 0

namespace PWM
{
    template<class T> inline std::string ToString(const T & value)
    {

```

```

        std::stringstream ss;
        ss << value;
        return ss.str();
    }

    inline void WriteToFile(const std::string & filePath, const std::string & value)
    {
#if DEBUG_VERBOSE_OUTPUT
        std::cout << "Writing: " << value << " to: " << filePath << std::endl;
#endif
        std::ofstream out;
        out.open(filePath.c_str());
        out.exceptions(std::ios::badbit);
        out << value << std::endl;
        out.close();
    }

// A bunch of helper functions to get us locations in the file system.
// They are used so that we can manipulate the pwm driver through the /sys
interface
    std::string GetFullNameOfFileInDirectory(const std::string & dirName, const
std::string & fileNameToFind);
    std::string GetCapeManagerSlotsPath();
    std::string GetOCPPath();
    int GetCapeManagerSlot(const std::string & moduleName);
    void LoadDeviceTreeModule(const std::string & name);
    void UnloadDeviceTreeModule(const std::string name);

class Pin
{
public:
    enum RunStatus
    {
        Free = -2, WaitingForSetup, Disabled, Enabled,
    };
    enum Polarity
    {
        PolarityHigh = 0, PolarityLow,
    };

private:
    std::string m_dutyFilePath;
    std::string m_periodFilePath;
    std::string m_polarityFilePath;
    std::string m_runFilePath;

```

```

        std::string m_pinName;

        long m_periodNS;
        long m_dutyNS;
        Polarity m_polarity;
        RunStatus m_runStatus;

public:
    const std::string &GetDutyFilePath() const
    {
        return m_dutyFilePath;
    }
    const std::string &GetPeriodFilePath() const
    {
        return m_periodFilePath;
    }
    const std::string &GetPolarityFilePath() const
    {
        return m_polarityFilePath;
    }
    const std::string &GetPinName() const
    {
        return m_pinName;
    }
    const std::string &GetRunFilePath() const
    {
        return m_runFilePath;
    }

    const RunStatus &GetRunStatus() const
    {
        return m_runStatus;
    }

    const long &GetPeriodNS() const
    {
        return m_periodNS;
    }
    const Polarity &GetPolarity() const
    {
        return m_polarity;
    }
    const long &GetDutyNS() const
    {
        return m_dutyNS;
    }
}

```

```

private:
    void WriteDutyNSToFile()
    {
        WriteToFile(GetDutyFilePath(), ToString(GetDutyNS()));
    }
    void WritePeriodNSToFile()
    {
        WriteToFile(GetPeriodFilePath(), ToString(GetPeriodNS()));
    }
    void WritePolarityToFile()
    {
        WriteToFile(GetPolarityFilePath(), GetPolarity() == PolarityHigh
? std::string("0") : std::string("1"));
    }

public:
    void SetDutyNS(const long & dutyNS)
    {
        m_dutyNS = std::min(dutyNS, GetPeriodNS());
        if (GetRunStatus() == Enabled)
            WriteDutyNSToFile();
    }
    void SetDutyUS(const int &dutyUS)
    {
        SetDutyNS((long) dutyUS *
MICRSECONDS_TO_NANOSECONDS);
    }
    void SetDutyMS(const int &dutyMS)
    {
        SetDutyNS((long) dutyMS *
MILLISECONDS_TO_NANOSECONDS);
    }
    void SetDutyPercent(const float &percent)
    {
        SetDutyNS(long(GetPeriodNS() * percent));
    }

    void SetPeriodNS(const long & periodNS)
    {
        if (GetRunStatus() == Enabled || GetRunStatus() == Disabled)
        {
            std::cout << "Trying to set the period but we need to
release the PWM module first!" << std::endl;
            throw std::bad_exception();
            return;
        }
    }
}

```

```

        }
        m_periodNS = periodNS;
        if (GetRunStatus() == Enabled)
            WritePeriodNSToFile();
    }
    void SetPeriodUS(const int &periodUS)
    {
        SetPeriodNS((long) periodUS *
MICRSECONDS_TO_NANOSECONDS);
    }
    void SetPeriodMS(const int &periodMS)
    {
        SetPeriodNS((long) periodMS *
MILLISECONDS_TO_NANOSECONDS);
    }

    void SetPolarity(const Polarity & polarity)
    {
        m_polarity = polarity;
        if (GetRunStatus() == Enabled)
            WritePolarityToFile();
    }

private:
    void SetRunStatus(const RunStatus & newRunStatus)
    {
        if (newRunStatus != GetRunStatus())
        {
            if (newRunStatus == Disabled)
            {
                WriteToFile(GetRunFilePath(), std::string("0"));
            }
            else if (newRunStatus == Enabled)
            {
                if (GetRunStatus() == Free)
                {
                    InitPinFS();
                }
                // Force write the file values out
                WritePeriodNSToFile();
                WriteDutyNSToFile();
                WritePolarityToFile();

                WriteToFile(GetRunFilePath(), std::string("1"));
            }
            else if (newRunStatus == Free)

```

```

        {
            if (GetRunStatus() != Disabled)
            {
                SetRunStatus(Disabled);
            }
            UnloadDeviceTreeModule(GetPinName());
        }
    }
    m_runStatus = newRunStatus;
}

public:
    void Enable()
    {
        SetRunStatus(Enabled);
    }
    void Disable()
    {
        SetRunStatus(Disabled);
    }
    void Release()
    {
        SetRunStatus(Free);
    }

public:
    ~Pin()
    {
        Release();
    }
    Pin(const std::string & pinName, const long & periodNS = 20 *
        MILLISECONDS_TO_NANOSECONDS, const long & dutyNS = 1 *
        MILLISECONDS_TO_NANOSECONDS) :
        m_pinName(pinName)
    {
        // If the pin is already in use then we need to free it!
        if (GetCapeManagerSlot(GetPinName()) != -1)
            UnloadDeviceTreeModule(GetPinName());

        m_runStatus = WaitingForSetup;

        SetPeriodNS(periodNS);
        SetDutyNS(dutyNS);
        SetPolarity(PolarityHigh);

        InitPinFS();
    }
}

```

```

void InitPinFS()
{
    LoadDeviceTreeModule(std::string("am33xx_pwm"));
    std::string pinModule = std::string("sc_pwm_") + GetPinName();
    LoadDeviceTreeModule(pinModule);
    std::string pinInterfacePath = GetOCPPPath() +
GetFullNameOfFileInDirectory(GetOCPPPath(), GetPinName()) + "/";
    m_dutyFilePath = pinInterfacePath + "duty";
    m_periodFilePath = pinInterfacePath + "period";
    m_polarityFilePath = pinInterfacePath + "polarity";
    m_runFilePath = pinInterfacePath + "run";

#if DEBUG_VERBOSE_OUTPUT
    std::cout << GetDutyFilePath() << std::endl;
    std::cout << GetPeriodFilePath() << std::endl;
    std::cout << GetPolarityFilePath() << std::endl;
    std::cout << GetRunFilePath() << std::endl;
#endif
}
};

#endif /* PWM_H_ */

```

root/src/BeagleBone/include/RealTime/Tasks/gpio.h

```

#ifndef GPIO_H_
#define GPIO_H_

#include <fcntl.h>
#include <unistd.h>
#include <cstdio>
#include <cstring>
#include <cstdlib>

#define MAX_BUF 64
#define GPIO_POLL_TIMEOUT 3000
#define FALSE 0           // False Macro
#define TRUE 1            // True Macro
#define SYSFS_GPIO_DIR "/sys/class/gpio"      // File System GPIO directory (note
that the firmware or .dtb file is loaded at startup (via .profile file) to enable

int gpio_export(unsigned int gpio);

```

```
int gpio_unexport(unsigned int gpio);
int gpio_set_dir(unsigned int gpio, unsigned int out_flag);
int gpio_set_value(unsigned int gpio, unsigned int value);
int gpio_get_value(unsigned int gpio, unsigned int *value);
int gpio_set_edge(unsigned int gpio, const char *edge);
int gpio_fd_open(unsigned int gpio);
int gpio_fd_close(int fd);

#endif
```

root/src/BeagleBone/include/RealTime/Tasks/AvoidCollisions/avoidcollisions.h

```
#ifndef AVOID_COLLISIONS_H_
#define AVOID_COLLISIONS_H_
```

```
void *avoid_collisions(void*);
```

```
#endif
```

root/src/BeagleBone/include/RealTime/Tasks/MonitorBluetooth/monitorbluetooth.h

```
#ifndef MONITOR_BLUETOOTH_H_
#define MONITOR_BLUETOOTH_H_
```

```
void openSerialPort(void);
void haltMonitorBluetooth(void);
void *monitor_bluetooth(void*);
```

```
#endif
```

root/src/BeagleBone/include/RealTime/Tasks/MonitorEncoders/monitorencoders.h

```
#ifndef MONITOR_ENCODERS_H_
#define MONITOR_ENCODERS_H_
```

```
void *monitor_encoders(void *period);
void cleanUpEncoders(void);
```

```
#endif
```

root/src/BeagleBone/include/RealTime/Tasks/MonitorEncoder/monitorencoder.h

```
#ifndef ENCODER_H_
#define ENCODER_H_
```

```
#include "gpio.h"
```

```

#include <poll.h>
#include <pthread.h>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <time.h>
#include <string.h>
#include <sys/time.h>
#include <ctime>
#include "time-math.h"
#include <errno.h>

using namespace std;
class Encoder {
    private:
        struct encoder_thread_p {
            unsigned volatile int encoder_count;
            pthread_t monitor_thread;
            unsigned int pinNum;
        } encoder_thread_data;
        float speed;

    public:
        Encoder(const void* );
        void *monitor_encoder(void * );
        static void *monitor_wrapper(void *context) {
            Encoder* eThis = static_cast<Encoder*>(context);
            eThis->monitor_encoder(context);
            delete eThis;
        }
        pthread_mutex_t encoder_mutex;
        void setSpeed(float newspeed) { speed = newspeed; }
        float getSpeed(void) { return speed; }
        int getCount(void) { return encoder_thread_data.encoder_count; }
        void resetCount(void) { encoder_thread_data.encoder_count = 0; }
        void start(Encoder * );
        pthread_t getMonitorThread(void) {
            return (this->encoder_thread_data).monitor_thread;
        }
    };
#endif

```

root/src/BeagleBone/include/RealTime/Tasks/MotorUpdater/motorupdater.h

```

#ifndef MOTOR_UPDATER_H_
#define MOTOR_UPDATER_H_

```

```

void *motor_updater(void*);
```

```
#endif
```

```
root/src/BeagleBone/include/RealTime/Tasks/LogData/logdata.h
```

```
#ifndef LOG_DATA_H_
#define LOG_DATA_H_
#include <string>
#include <iostream>
#include <queue>
```

```
void createLogFile(void);
void write_text_to_log_file(const std::string &text);
void *log_data(void *);
```

```
#endif
```

```
root/src/BeagleBone/include/RRT/RRT_class.h
```

```
/*
 * RRT_class.h
 *
 * Created on: Apr 16, 2014
 * Author: Thomas
 */
```

```
#ifndef RRT_CLASS_H_
#define RRT_CLASS_H_
#define _USE_MATH_DEFINES
```

```
#define ENABLE_ABA           // Active brake assist
// #define STORE_FAILED_PATHS // Store failed paths
```

```
#include <cmath>
#include <vector>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <string.h>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
class RRT {
```

```

typedef vector< vector<double> > ArrayList;
typedef vector<double> ArrayDbl;
typedef vector<int> ArrayInt;

enum _sensor_type { LIDAR, ULTASONIC };

class ABA {

    private:
        int brake_percent;
        bool ABA_event;
        double prev_distABA;
        double curr_distABA;
        double Vrel;
        double TTC;
        double target;
        bool updateFirstDistFlag;
        float GAIN;
        _sensor_type sensor_type;

    public:
        ABA(double, _sensor_type);
        void zero();
        void update(double, ArrayList, double);
        void update(double, double, double);
        bool getABAEvent() { return ABA_event; };
        int getABABrake() { return brake_percent; };
    };

    private:

        double Road_Lines[3];
        double Goal[2];
        double Temp_Node[3];
        int pathCount;

        double Path[50][5];
        double Final_Path[50][5];
        double Route_Tree[400][2];
        double Prev_Path[50][5];
        vector< vector< vector<double> > > failedPaths;

        void Get_Roadlines(ArrayDbl);
        bool roadDetected(ArrayDbl);
        void Get_Goal(ArrayDbl);
        void AddLinear(float,ArrayList,int);

```

```

    void AddRandom(float,ArrayList,int,ArrayDbl);
    double unifRandNumber();
    void Get_Path(ArrayList,float,ArrayDbl,float[]);
    bool CheckCollision(ArrayDbl, float[],ArrayDbl);
    bool enableABA;
    double minTimeToCollision;
    ABA *aba;

public:
    RRT();
    ~RRT();
    float getRRTSteer();
    void Run_RRT(ArrayDbl,ArrayList,ArrayDbl);
    ArrayList getFinalPath();
    vector< vector< vector<double> > > getFailedPaths();
    bool getRRTVerify();
    bool getABAEvent() { return aba->getABAEvent(); };
    int getABABrake() { return aba->getABABrake(); };

};

#endif /* RRT_CLASS_H_ */

```

root/src/BeagleBone/include/Segmentation/segmentation\_class.h

```

/*
 * segmentation_class.h
 *
 * Created on: Apr 16, 2014
 *      Author: Thomas
 */

#ifndef SEGMENTATION_CLASS_H_
#define SEGMENTATION_CLASS_H_
#define _USE_MATH_DEFINES

#define ENABLE_LDW          // Lane-departure warning
#define ENABLE_LKA           // Lane-keep assist

#include <cmath>
#include <iostream>
#include <vector>
#include <algorithm>
#include <string.h>
using namespace std;

```

```

enum threat_type_t { NONE, OBSTACLE, ROADBOUNDARY, RRT_MANUEVER };
enum threat_region_t { NA, LOW, MEDIUM, HIGH };

class Segmentation {
    typedef vector< vector<double> > ArrayList;
    typedef vector<int> ArrayInt;
    typedef vector<double> ArrayDbl;
private:
    ArrayList    AllObstacles;
    ArrayList    ReturnObstacles;
    ArrayList    ReturnAllObstacles;
    ArrayList    ThreateningObstacles;
    ArrayDbl     ThreateningMarkers;
    ArrayList    ReturnThreateningMarkers;
    ArrayList    Obstacle;
    ArrayList    RoadPts;
    ArrayInt     PossibleRoadPts;
    ArrayInt     toRemove;

    threat_type_t threat_type;
    threat_region_t threat_region;
    int      Threshold;
    double   maxNumPts;
    double   ScalingFactor;
    double   MaxVelocity;
    double   angularRes_rad;
    double   areamin;
    int      noRoadCount;
    bool    noRoadFlag;
    bool    feasibleRoadway;
    double  roadBounds[6];
    double  prevRoadBounds[6];
    double  prevRoadSlope;
    double  deltaYaw;
    double  currentYaw;
    double  prevYaw;
    double  rotateIndices;
    double  maxSlopeChange;
    double  areaThreshold;
    double  roadWidth;
    double  areaBuffer;
    double  coneSpacing;
    double  spacingBuffer;
    double  coneRoadBoundOffset;
    double  threatThreshold;
    double  distThreshold;
}

```

```

int      prevToggle;
int      Toggle;
double minSpeed;
double minDistance;
double threatConeAngle;
double allowableDistToCone;
double emergencyDistToCone;
double thresholdManueverAngle;
double maxStepSize;
int      BEAMNUMBER;
bool    roadBoundaryYawThreat_possible;

double getBoxLength(double mini,double maxi,double mLength) {
    double dist=abs(maxi-mini)*ScalingFactor;
    if(dist<mLength)
        dist = mLength;
    return dist;
};

double chkMaxVelocity(double input,double accumulated,int
iterationNum) {
    if(std::abs(input)>MaxVelocity)
        return 0;
    else
        return input;
};

double* calculateSlopeIntercept(double[],double[]);
bool checkRoadSlope(double);
void segmentData();
void determineThreat(double[]);
void getObstacleInfo(double,double);
void updateRoadBounds();
void clusterData(double[],double[],double[],double[],double);
void parseRoadBounds();
void removeRoadCones();
void cleanUpObstacles();
void getCorrectObstacleInfo(double,double);

public:
    Segmentation();
    Segmentation(int, int);
    Segmentation(int,
    int,double,double,double,double,double,double,double,double,double,
    double,double,double);
    Segmentation(int,
    int,double,double,double,double,double,double,double,double,double,
    double,double,double,int);

```

```

void
updateCorrectAdjustedVelocities(double[],double[],double[],double[],double);
    void identifyObstacles(double [],double[],double[],double[],double);
    void identifyObstacles(double[],double[],double[],double[],double,
double[],int);
        threat_type_t getThreatType();
        threat_region_t getThreatRegion();
        ArrayList getObstacleArray();
        ArrayList getAllObstacleArray();
        ArrayDbl returnRoadBounds() {      ArrayDbl returnBounds;
returnBounds.push_back(roadBounds[0]); returnBounds.push_back(roadBounds[1]);
returnBounds.push_back(roadBounds[2]);
                                         return
returnBounds;
    }
    int returnThreat() { return Toggle; }
    //~Segmentation();
};

#endif /* SEGMENTATION_CLASS_H_ */

```

root/src/BeagleBone/source/RealTime/BluetoothDriver\_RT.cpp

```
root/src/BeagleBone/source/RealTime/BluetoothDriver_RT.cpp

/** \file BluetoothDriver_RT.cpp
 * \brief Soft real-time Linux based bluetooth driver firmware
 *
 */
*****  
* Library Imports  
*****  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include <stdint.h>  
#include <signal.h>  
  
#include <unistd.h>  
#include <iostream>  
#include <stdlib.h>  
#include <errno.h>  
#include <time.h>  
  
#include <ctime>  
#include <string>
```

```

#include <poll.h>
#include <pthread.h>
#include <cstdio>
#include <sys/time.h>

#include "globaldata.h"
#include "monitorbluetooth.h"
#include "motorupdater.h"
#include "logdata.h"
#include "PWM.h"
/*****
* Constants
*****/
#define _POSIX_SOURCE 1      // POSIX compliant source
/*****
* Global Variables/Tasks/Threads
*****/
volatile bool globalRUN = true;
pthread_mutex_t global_RUN_mutex;
pthread_t poll_bluetooth;
pthread_t update_motors;
pthread_t log_data_task;
long poll_bluetooth_period = 50000L; // poll speed every 50ms = 50Hz
long motor_update_period = 10000000L; // PWM freq
long log_data_period = 50000L;

int logOption = 0;
int fullAuto = 0;
int holdThrottle = 0;
int indicateFreq = 0;
volatile int validPath = 0;
volatile float avgSpeed = 0.0;
volatile int threat = 0;

using namespace std;
/*****
* ^C Interrupt Handler
*****/
void cntrlCHandler(int s) {
    pthread_mutex_lock(&global_RUN_mutex);
    globalRUN = false;
    pthread_mutex_unlock(&global_RUN_mutex);
    int chk = pthread_join( poll_bluetooth, NULL);
    pthread_cancel(update_motors);
}

```

```

void printHelp() {
    cout<<"HELP"<<endl<<endl;
    cout<<"This program receives the steer and throttle command from the user control
station as two bytes. ";
    cout<<endl<<"The bluetooth module device and baudrate can be set by the user
(described below)"<<endl;
    cout<<"If a 0,0 if received an emergency stop state is toggled and can be untoggled
by receiving another"<<endl;
    cout<<"0,0. In addition a timeout, adjustabled by user as well, is used to trigger a
neutral state if no "<<endl;
    cout<<"input commands are received."<<endl<<endl;
    cout<<"-b <baudrate> or --baudrate <baudrate>"<<endl<<" Sets the
baudrate"<<endl<<endl;
    cout<<"-D <device> or --device <device>"<<endl<<" Set the device
port"<<endl<<endl;
    cout<<"-t <timeout> or --timeout <timeout>"<<endl<<" Sets the timeout till
neutral state"<<endl;
}
/***********************
* Main
***********************/
int main(int argc, char **argv, char **envp)
{
    // Read program arguments
    if(argc > 1) {
        for(int i = 1; i < argc; i+=2) {
            string flag = argv[i];
            if((flag == "-h")||(flag == "--help")) {
                printHelp();
                exit(1);
            }
            if((flag == "-b")||(flag == "--baudrate")) {
                // Set baudrate
                switch(atoi(argv[i+1])) {
                    case 9600:
                        baudrate = B9600;
                        break;
                    case 115200:
                        baudrate = B115200;
                        break;
                }
            }
            if((flag == "-D")||(flag == "--device")) {
                // Set serial port
                modemdevice = argv[i+1];
            }
        }
    }
}

```

```

        if((flag == "-t")||(flag == "--timeout")) {
            // update timeout
            poll_timeout = atof(argv[i+1]);
        }
    }
}

//Register Cntrl C Interrupt Handler
struct sigaction sigIntHandler;
sigIntHandler.sa_handler = cntrlCHandler;
sigemptyset(&sigIntHandler.sa_mask);
sigIntHandler.sa_flags = 0;

sigaction(SIGINT, &sigIntHandler, NULL);

pthread_attr_t my_attr;
struct sched_param param;

pthread_attr_init(&my_attr);
pthread_attr_setinheritsched(&my_attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&my_attr, SCHED_RR );

int chk;
// Initialze the encoder mutex (mutual exclusion)
chk = pthread_mutex_init(&global_RUN_mutex, NULL);
if(chk) {
    perror("pthread_mutex create failed\n");
    pthread_exit(NULL);
}

//Initialze poll_blueooth task/thread
param.sched_priority = 40;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &update_motors, &my_attr, motor_updater,
&motor_update_period );
if(chk) { cout << "unable to create motor updater thread" << endl; }

param.sched_priority = 60;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &poll_bluetooth, &my_attr, monitor_bluetooth,
&poll_bluetooth_period );
if(chk) { cout << "unable to create poll bluetooth thread" << endl; }

if((logOption == 1)||(logOption == 3)) {
param.sched_priority = 1;
pthread_attr_setschedparam(&my_attr, &param);
}

```

```

chk = pthread_create( &log_data_task, &my_attr, log_data, &log_data_period );
if(chk) { cout << "unable to create log data thread" << endl; }
}

pthread_attr_destroy(&my_attr);

// Exit main program
cout<<endl<<"Exiting main program"<<endl;
pthread_exit(NULL); /* exit main thread */

return 0;
}

```

root/src/BeagleBone/source/RealTime/BluetoothDriverwithSpeed\_RT.cpp

```

/**      \file BluetoothDriver_RT.cpp
*       \brief Soft real-time Linux based bluetooth driver firmware
*
*/
***** * Library Imports *****
*****/



#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdint.h>
#include <signal.h>

#include <unistd.h>
#include <iostream>
#include <stdlib.h>
#include <errno.h>
#include <time.h>

#include <ctime>
#include <string>

#include <poll.h>
#include <pthread.h>
#include <cstdio>
#include <sys/time.h>

#include "globaldata.h"
#include "logdata.h"

```

```

#include "monitorbluetooth.h"
#include "monitorencoders.h"
#include "motorupdater.h"
#include "PWM.h"
/*****
* Constants
*****
#define _POSIX_SOURCE 1      // POSIX compliant source
/*****
* Global Variables/Tasks/Threads
*****
volatile bool globalRUN = true;
pthread_mutex_t global_RUN_mutex;
pthread_t poll_bluetooth;
pthread_t update_motors;
pthread_t poll_encoders;
pthread_t log_data_task;
long poll_bluetooth_period = 50000L; // poll speed every 50ms = 20Hz
long motor_update_period = 10000000L; // PWM freq
long poll_encoders_period = 50000L; // poll speed every 50ms = 20Hz
long log_data_period = 50000L;

int logOption = 0;
int fullAuto = 0;
int holdThrottle = 0;
volatile int validPath = 0;
volatile int threat = 0;

using namespace std;
/*****
* ^C Interrupt Handler
*****
void cntrlCHandler(int s) {
    pthread_mutex_lock(&global_RUN_mutex);
    globalRUN = false;
    pthread_mutex_unlock(&global_RUN_mutex);
    int chk = pthread_join( poll_bluetooth, NULL);
    pthread_cancel(update_motors);
}
void printHelp() {
    cout<<"HELP"<<endl<<endl;
    cout<<"This program receives the steer and throttle command from the user control
station as two bytes. ";
    cout<<endl<<"The bluetooth module device and baudrate can be set by the user
(described below)"<<endl;

```

```

cout<<"If a 0,0 if received an emergency stop state is toggled and can be untoggled
by receiving another"<<endl;
cout<<" 0,0. In addition a timeout, adjustabled by user as well, is used to trigger a
neutral state if no "<<endl;
cout<<"input commands are received."<<endl<<endl;
cout<<"-b <baudrate> or --baudrate <baudrate>"<<endl<<"  Sets the
baudrate"<<endl<<endl;
cout<<"-D <device> or --device <device>"<<endl<<"  Set the device
port"<<endl<<endl;
cout<<"-t <timeout> or --timeout <timeout>"<<endl<<"  Sets the timeout till
neutral state"<<endl;
}
/***********************
* Main
***********************/
int main(int argc, char **argv, char **envp)
{
    // Read program arguments
    if(argc > 1) {
        for(int i = 1; i < argc; i+=2) {
            string flag = argv[i];
            if((flag == "-h")||(flag == "--help")) {
                printHelp();
                exit(1);
            }
            if((flag == "-b")||(flag == "--baudrate")) {
                // Set baudrate
                switch(atoi(argv[i+1])) {
                    case 9600:
                        baudrate = B9600;
                        break;
                    case 115200:
                        baudrate = B115200;
                        break;
                }
            }
            if((flag == "-D")||(flag == "--device")) {
                // Set serial port
                modemdevice = argv[i+1];
            }
            if((flag == "-t")||(flag == "--timeout")) {
                // update timeout
                poll_timeout = atof(argv[i+1]);
            }
            if((flag == "-l")||(flag == "--log")) {
                logOption = atoi(argv[i+1]);
            }
        }
    }
}
```

```

        }
    }
}

//Register Cntrl C Interrupt Handler
struct sigaction sigIntHandler;
sigIntHandler.sa_handler = cntrlCHandler;
sigemptyset(&sigIntHandler.sa_mask);
sigIntHandler.sa_flags = 0;

sigaction(SIGINT, &sigIntHandler, NULL);

pthread_attr_t my_attr;
struct sched_param param;

pthread_attr_init(&my_attr);
pthread_attr_setinheritsched(&my_attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&my_attr, SCHED_RR);

int chk;
// Initialize the encoder mutex (mutual exclusion)
chk = pthread_mutex_init(&global_RUN_mutex, NULL);
if(chk) {
    perror("pthread_mutex create failed\n");
    pthread_exit(NULL);
}

//Initialize poll_blueooth task/thread
//Initialze poll_encoder task/thread
param.sched_priority = 5;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &poll_encoders, &my_attr, monitor_encoders,
&poll_encoders_period );
if(chk) { cout << "unable to create poll encoders thread" << endl; }

param.sched_priority = 40;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &update_motors, &my_attr, motor_updater,
&motor_update_period );
if(chk) { cout << "unable to create motor updater thread" << endl; }

param.sched_priority = 60;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &poll_bluetooth, &my_attr, monitor_bluetooth,
&poll_bluetooth_period );
if(chk) { cout << "unable to create poll bluetooth thread" << endl; }

```

```

        if((logOption == 1)||(logOption == 3)) {
            param.sched_priority = 1;
            pthread_attr_setschedparam(&my_attr, &param);
            chk = pthread_create( &log_data_task, &my_attr, log_data, &log_data_period );
            if(chk) { cout << "unable to create log data thread" << endl; }
        }

        pthread_attr_destroy(&my_attr);
        // Exit main program
        cout<<endl<<"Exiting main program"<<endl;
        pthread_exit(NULL); /* exit main thread */

    return 0;
}

```

root/src/BeagleBone/source/RealTime/FullSys\_RT.cpp

```

/**      \file FullSys_RT.cpp
 *       \brief Soft real-time Linux based collision avoidance firmware for embedded
hardware
*
*/
/// \todo CHECK STACK SIZES
/********************* */
* Library Imports
/********************* /
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdint.h>
#include <signal.h>
#include <time.h>

#include <unistd.h>
#include <iostream>
#include <stdlib.h>
#include <errno.h>

#include <ctime>
#include <string>

#include <poll.h>
#include <pthread.h>

```

```

#include <cstdio>
#include <sys/time.h>

#include "globaldata.h"
#include "logdata.h"
#include "avoidcollisions.h"
#include "monitorencoders.h"
#include "monitorbluetooth.h"
#include "motorupdater.h"
#include "PWM.h"
/*****
* Constants
*****
#define _POSIX_SOURCE 1 /* POSIX compliant source */
/*****
* Global Variables/Tasks/Threads
*****
volatile bool globalRUN = true;
pthread_mutex_t global_RUN_mutex;

pthread_t poll_encoders;
long poll_encoders_period = 50000L; // poll speed every 50ms = 20Hz

pthread_t update_motors;
long motor_update_period = 10000000L; // PWM freq

pthread_t poll_bluetooth;
long poll_bluetooth_period = 50000L;

pthread_t collision_avoidance;
long collision_avoidance_period = 50000L;

pthread_t log_data_task;
long log_data_period = 1000000L;

int logOption = 0;
int fullAuto = 0;
int holdThrottle = 0;
int indicateFreq = 0;

using namespace std;
/*****
* ^C Interrupt Handler
*****
void cntrlCHandler(int s) {
    pthread_mutex_lock(&global_RUN_mutex);
}

```

```

        globalRUN = false;
        pthread_mutex_unlock(&global_RUN_mutex);
        int chk = pthread_join( poll_bluetooth, NULL);
        pthread_cancel(update_motors);
    }
/*********************************************************/
* Print Help Menu
/*********************************************************/
void printHelp() {
    cout<<"HELP"<<endl<<endl;
    cout<<"This program receives the steer and throttle command from the user control
station as two bytes. ";
    cout<<endl<<"The bluetooth module device and baudrate can be set by the user
(described below)"<<endl;
    cout<<"If a 0,0 if received an emergency stop state is toggled and can be untoggled by
receiving another"<<endl;
    cout<<" 0,0. In addition a timeout, adjustabled by user as well, is used to trigger a
neutral state if no "<<endl;
    cout<<"input commands are received."<<endl<<endl;
    cout<<"-b <baudrate> or --baudrate <baudrate>"<<endl<<"  Sets the
baudrate"<<endl<<endl;
    cout<<"-D <device> or --device <device>"<<endl<<"  Set the device
port"<<endl<<endl;
    cout<<"-t <timeout> or --timeout <timeout>"<<endl<<"  Sets the timeout till the
neutral state is set after "<<endl<<"not receiving user commands"<<endl;
    cout<<"-l <config> or --log <config>"<<endl<<"  Sets the log
configuration:<<endl<<"  1=Record only speed and control signals"<<endl<<
2=Record only LiDAR point-cloud data"<<endl;
    cout<<"  3=Record both speed & control signals and the LiDAR data"<<endl;
    cout<<"-a or --auto"<<endl<<"  Sets the collision avoidance to act fully
autonomously. In this mode the user only controls the throttle command"<<endl;
    cout<<"-H or --hold"<<endl<<"  Sets the system to maintain the vehicle's speed
when the system is enabled"<<endl;
    cout<<"-i or --indicate"<<endl<<"  Sets the LED to flash at the frequency that
the segmentation, threat determination, and path planning occurs"<<endl;
}
/*********************************************************/
* Main
/*********************************************************/
int main(int argc, char **argv, char **envp)
{
    // Read program arguments
    if(argc > 1) {
        for(int i = 1; i < argc; i+=2) {
            string flag = argv[i];
            if((flag == "-h")||(flag == "--help")) {

```

```

printHelp();
exit(1);
}
if((flag == "-b")||(flag == "--baudrate")) {
    // Set baudrate
    switch(atoi(argv[i+1])) {
        case 9600:
            baudrate = B9600;
            break;
        case 115200:
            baudrate = B115200;
            break;
    }
}
if((flag == "-D")||(flag == "--device")) {
    // Set serial port
    modemdevice = argv[i+1];
}
if((flag == "-t")||(flag == "--timeout")) {
    // update timeout
    poll_timeout = atof(argv[i+1]);
}
if((flag == "-l")||(flag == "--log")) {
    logOption = atoi(argv[i+1]);
}
if((flag == "-a")||(flag == "--auto")) {
    fullAuto = 1;
}
if((flag == "-H")||(flag == "--hold")) {
    holdThrottle = 1;
}
if((flag == "-i")||(flag == "--indicate")) {
    indicateFreq = 1;
}
}
}
} else {
    //exit
    printHelp();
    exit(1);
}

// Register Cntrl C Interrupt Handler
struct sigaction sigIntHandler;
sigIntHandler.sa_handler = cntrlCHandler;
sigemptyset(&sigIntHandler.sa_mask);
sigIntHandler.sa_flags = 0;

```

```

sigaction(SIGINT, &sigIntHandler, NULL);

pthread_attr_t my_attr;
struct sched_param param;

pthread_attr_init(&my_attr);
pthread_attr_setinheritsched(&my_attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&my_attr, SCHED_RR );

int chk;
// Initialize the encoder mutex (mutual exclusion)
chk = pthread_mutex_init(&global_RUN_mutex, NULL);
if(chk) {
    perror("pthread_mutex create failed\n");
    pthread_exit(NULL);
}

// Initialize poll_encoder task/thread
param.sched_priority = 5;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &poll_encoders, &my_attr, monitor_encoders,
&poll_encoders_period );
if(chk) { cout << "unable to create poll encoders thread" << endl; }

// Initialize update_motors task/thread
param.sched_priority = 40;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &update_motors, &my_attr, motor_updater,
&motor_update_period );
if(chk) { cout << "unable to create motor updater thread" << endl; }

// Initialize poll_blueooth task/thread
param.sched_priority = 60;
pthread_attr_setschedparam(&my_attr, &param);
chk = pthread_create( &poll_bluetooth, &my_attr, monitor_bluetooth,
&poll_bluetooth_period );
if(chk) { cout << "unable to create poll bluetooth thread" << endl; }

// Initialize collision_avoidance task/thread
param.sched_priority = 80;
pthread_attr_setschedparam(&my_attr, &param);
pthread_create( &collision_avoidance, &my_attr, avoid_collisions,
&collision_avoidance_period );
if(chk) { cout << "unable to create collision avoidance thread" << endl; }

```

```

        if((logOption == 1)||(logOption == 3)) {
            param.sched_priority = 1;
            pthread_attr_setschedparam(&my_attr, &param);
            chk = pthread_create( &log_data_task, &my_attr, log_data,
&log_data_period );
            if(chk) { cout << "unable to create log data thread" << endl; }
        }

        pthread_attr_destroy(&my_attr);

        // Exit main program
        cout<<endl<<"Exiting main program"<<endl;
        pthread_exit(NULL); /* exit main thread */

        return 0;
    }

```

root/src/BeagleBone/source/RealTime/Tasks/gpio.cpp

```

#include "gpio.h"

/*****************
 * gpio_export
 *****************/
int gpio_export(unsigned int gpio)
{
    int fd, len;
    char buf[MAX_BUF];

    fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
    if (fd < 0) {
        perror("gpio/export");
        return fd;
    }

    len = snprintf(buf, sizeof(buf), "%d", gpio);
    write(fd, buf, len);
    close(fd);

    return 0;
}

/*****************
 * gpio_unexport
 *****************/
int gpio_unexport(unsigned int gpio)

```

```

{
    int fd, len;
    char buf[MAX_BUF];

    fd = open(SYSFS_GPIO_DIR "/unexport", O_WRONLY);
    if (fd < 0) {
        perror("gpio/export");
        return fd;
    }

    len = sprintf(buf, sizeof(buf), "%d", gpio);
    write(fd, buf, len);
    close(fd);
    return 0;
}
/*********************************************************/
* gpio_set_dir
/*********************************************************/
int gpio_set_dir(unsigned int gpio, unsigned int out_flag)
{
    int fd;
    char buf[MAX_BUF];

    sprintf(buf, sizeof(buf), SYSFS_GPIO_DIR " /gpio%d/direction", gpio);

    fd = open(buf, O_WRONLY);
    if (fd < 0) {
        perror("gpio/direction");
        return fd;
    }

    if (out_flag)
        write(fd, "out", 4);
    else
        write(fd, "in", 3);

    close(fd);
    return 0;
}

/*********************************************************/
* gpio_set_value
/*********************************************************/
int gpio_set_value(unsigned int gpio, unsigned int value)
{
    int fd;

```

```

char buf[MAX_BUF];

snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

fd = open(buf, O_WRONLY);
if (fd < 0) {
    perror("gpio/set-value");
    return fd;
}

if (value)
    write(fd, "1", 2);
else
    write(fd, "0", 2);

close(fd);
return 0;
}

/*****************/
* gpio_get_value
/*****************/
int gpio_get_value(unsigned int gpio, unsigned int *value)
{
    int fd;
    char buf[MAX_BUF];
    char ch;

    snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

    fd = open(buf, O_RDONLY);
    if (fd < 0) {
        perror("gpio/get-value");
        return fd;
    }

    read(fd, &ch, 1);

    if (ch != '0') {
        *value = 1;
    } else {
        *value = 0;
    }

    close(fd);
    return 0;
}

```

```

}

/*****
 * gpio_set_edge
 *****/
int gpio_set_edge(unsigned int gpio, const char *edge)
{
    int fd;
    char buf[MAX_BUF];

    snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/edge", gpio);

    fd = open(buf, O_WRONLY);
    if (fd < 0) {
        perror("gpio/set-edge");
        return fd;
    }

    write(fd, edge, strlen(edge) + 1);
    close(fd);
    return 0;
}

/*****
 * gpio_fd_open
 *****/
int gpio_fd_open(unsigned int gpio)
{
    int fd;
    char buf[MAX_BUF];

    snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);

    fd = open(buf, O_RDONLY | O_NONBLOCK );
    if (fd < 0) {
        perror("gpio/fd_open");
    }
    return fd;
}

/*****
 * gpio_fd_close
 *****/
int gpio_fd_close(int fd)
{
    return close(fd);
}

```

```
}
```

```
root/src/BeagleBone/source/RealTime/Tasks/PWM.cpp
```

```
/*
 * PWM.cpp
 *
 * Created on: May 30, 2013
 *      Author: Saad Ahmad ( http://www.saadahmad.ca )
 */
#include "PWM.h"
#include <unistd.h>

namespace PWM
{
    // A bunch of helper functions to get us locations in the file system.
    // They are used so that we can manipulate the pwm driver through the /sys
    interface
        std::string GetFullNameOfFileInDirectory(const std::string & dirName, const
        std::string & fileNameToFind)
    {
        DIR *pDir;

        dirent *pFile;
        if ((pDir = opendir(dirName.c_str())) == NULL)
        {
            std::cout << "Directory name: " << dirName << " doesnt exist!" <<
        std::endl;
            throw std::bad_exception();
        }
        while ((pFile = readdir(pDir)) != NULL)
        {
            std::string currentFileName = (pFile->d_name);
            if (currentFileName.find(fileNameToFind) != std::string::npos)
            {
                return currentFileName;
            }
        }
        return std::string("");
    }

    std::string GetCapeManagerSlotsPath()
    {
        static std::string g_capeManagerSlotsPath;
        if (g_capeManagerSlotsPath.length() <= 0)
        {
```

```

#if DEBUG_VERBOSE_OUTPUT
    std::cout << "Setting up cape manager path" << std::endl;
#endif
    std::string capebasePath("/sys/devices/");
    std::string fileName =
GetFullNameOfFileInDirectory(capebasePath, std::string("bone_capemgr."));
    g_capeManagerSlotsPath = capebasePath + fileName + "/slots";
}
return g_capeManagerSlotsPath;
}

std::string GetOCPPath()
{
    static std::string g_ocpPath;
    if (g_ocpPath.length() == 0)
    {
        std::string ocpbasePath("/sys/devices/");
        std::string ocpName =
GetFullNameOfFileInDirectory(ocpbasePath, std::string("ocp."));
        g_ocpPath = ocpbasePath + ocpName + '/';
    }
    return g_ocpPath;
}

int GetCapeManagerSlot(const std::string & moduleName)
{
#endif DEBUG_VERBOSE_OUTPUT
    std::cout << "Trying to find slot for module: " << moduleName <<
std::endl;
#endif
    std::ifstream in(GetCapeManagerSlotsPath().c_str());
    in.exceptions(std::ios::badbit);
    int slot = -1;
    while (in >> slot)
    {
        std::string restOfLine;
        std::getline(in, restOfLine);
        if (restOfLine.find(moduleName) != std::string::npos)
        {
#endif DEBUG_VERBOSE_OUTPUT
            std::cout << "Found Module: " << moduleName << " at
slot: " << slot << std::endl;
#endif
            return slot;
        }
    }
}

```

```

#if DEBUG_VERBOSE_OUTPUT
    std::cout << "Module: " << moduleName << " not found in cape
manager!" << std::endl;
#endif
    return -1;
}
void LoadDeviceTreeModule(const std::string & name)
{
    int slot = GetCapeManagerSlot(name);
    if (slot == -1)
    {
#ifndef DEBUG_VERBOSE_OUTPUT
        std::cout << "Adding Module: " << name << std::endl;
        std::cout << "Its going in: " << GetCapeManagerSlotsPath() <<
std::endl;
#endif
        WriteToFile(GetCapeManagerSlotsPath(), name);

        usleep(MODULE_DELAY_TIME_US);
    }
    else
    {
#ifndef DEBUG_VERBOSE_OUTPUT
        std::cout << "Module " << name << " is already in here!" <<
std::endl;
#endif
    }
}
void UnloadDeviceTreeModule(const std::string name)
{
    int currentSlot = GetCapeManagerSlot(name);
    if (currentSlot == -1)
    {
        std::cout << "Why is the module " << name << " being unloaded
when its not in use?" << std::endl;
        throw std::bad_exception();
    }
#ifndef DEBUG_VERBOSE_OUTPUT
    std::cout << "Unloading module: " << name << std::endl;
#endif
    WriteToFile(GetCapeManagerSlotsPath(), std::string("-") +
ToString(currentSlot));

    usleep(MODULE_DELAY_TIME_US);
}

```

root/src/BeagleBone/source/RealTime/Tasks/AvoidCollisions/avoidcollisions.cpp

```
#include "avoidcollisions.h"

#include "Urg_driver.h"
#include "Connection_information.h"
#include "math_utilities.h"
#include "time-math.h"
#include "../Segmentation/segmentation_class.h"
#include "../RRT/RRT_class.h"
#include "globaldata.h"
#include "gpio.h"
#include "constants.h"

#include <iostream>
#include <math.h>
#include <vector>
#include <pthread.h>
#include <cstdio>
#include <fstream>

using namespace std;
using namespace qrk;

typedef vector< vector<double> > ArrayList;
typedef vector<double> ArrayDbl;
typedef vector<int> ArrayInt;

volatile int threat = 0;
pthread_mutex_t threat_mutex;
volatile int validPath = 0;

double *pointCloudX;
double *pointCloudY;

ofstream ptCloudFile;
double elaspedTime = 0.0;

double followingTime = 1;

// Emergency Braking Variables
double distEBS;
double prev_distEBS;

int getPointCloud(const Urg_driver &urg, const vector<long> &data, long time_stamp) {
```

```

        long min_dist = urg.min_distance();
        long max_dist = urg.max_distance();
        size_t data_n = data.size();

        pointCloudX = new (nothrow) double[data_n];
        pointCloudY = new (nothrow) double[data_n];

        ostringstream convert;
        if(logOption > 1) {
            char timeString[20];
            time_t current_time = time(NULL);
            strftime(timeString, sizeof(timeString), "%H:%M:%S",
            localtime(&current_time));
            std::string time_String(timeString);
            ptCloudFile << time_String << " ";
            convert << elaspedTime << endl;
        }

        prev_distEBS = distEBS;
        distEBS = max_dist;

        for(size_t i = 0; i < data_n; ++i) {
            long l = data[i];
            if((l < min_dist) || (l > max_dist))
                l = max_dist;

            double radian = urg.index2rad(i);
            long x = static_cast<long>(l * cos(radian));
            long y = static_cast<long>(l * sin(radian));

            if((abs(x)*0.00328 <= wheelBaseWidth/2) && (y > 0)) {
                if(l*0.00328 < distEBS)
                    distEBS = l*0.00328;
            }

            pointCloudX[i] = -y*0.00328;
            pointCloudY[i] = x*0.00328;
            if(logOption > 1)
                convert << pointCloudX[i] << " " << pointCloudY[i] << endl;
        }
        if(logOption > 1)
            ptCloudFile << convert.str();

        return (int)data_n;
    }
    ****

```

```

* Avoid Collisions Task
*****
void *avoid_collisions(void *period) {
    // Declare variables on thread's stack memory
    long ps = *((long *) period);
    struct timespec requestStart, requestEnd;
    cout<<"Created Collision Avoidance Thread" << endl;
    int chk;
    // Initialize the encoder mutex (mutual exclusion)
    chk = pthread_mutex_init(&threat_mutex, NULL);
    if(chk) {
        perror("pthread_mutex create failed\n");
        pthread_exit(NULL);
    }
    // Segmentation/Parsing/Threat Class
    Segmentation *segmentation = new Segmentation();
    // Path-Planning Class
    RRT *rrt = new RRT();

    Connection_information information(0,NULL);
    Urg_driver *urg = new Urg_driver();

    if(!urg->open(information.device_or_ip_name(),
                    information.baudrate_or_port_number(),
                    information.connection_type())) {
        #ifdef DEBUG
            cout<<"Urg_driver::open():" << endl;
        #endif
        cout<<information.device_or_ip_name() << ": " << urg->what() << endl;
        perror("Unable to open URG sensor\n");
        pthread_mutex_lock(&global_RUN_mutex);
        globalRUN = false;
        pthread_mutex_unlock(&global_RUN_mutex);
    }

    urg->set_timeout_msec(100);
    urg->start_measurement(Urg_driver::Distance, Urg_driver::Infinity_times, 0);

    if(logOption > 1) {
        char timeString[20];
        time_t current_time = time(NULL);
        std::string basePath ("/media/ptcloud_");
        strftime(timeString, sizeof(timeString), "%Y-%m-%d_%H-%M-%S",
localtime(&current_time));
        std::string time_String(timeString);

```

```

std::string ptfilepath = baseFilePath+time_String;
#ifndef DEBUG
    cout<<"Logging point-cloud to "<<ptfilepath<<endl;
#endif
ptCloudFile.open(ptfilepath.c_str());
}

distEBS = urg->max_distance();
prev_distEBS = distEBS;
double diff = 0.0;

while(globalRUN) {
    clock_gettime(CLOCK_REALTIME,&requestStart);

    if(indicateFreq)
        gpio_set_value(LED2,1);

    vector<long> data;
    long time_stamp = 0;

    if(urg->get_distance(data,&time_stamp)) {

        int beams = getPointCloud(*urg, data, time_stamp);

        double vehicleSpeeds[] = {0, avgSpeed*1.466667};
        double vRel = (prev_distEBS-distEBS)/(avgSpeed*1.466667);
        double timeToCollision = distEBS/(vehicleSpeeds[1]-vRel);
        if(timeToCollision <= followingTime) {
            // EBS braking trigger

            // set threat high

        }
        /// \todo save previous point-cloud and pass in for obstacle
tracking/velocity estimation
        segmentation-
>identifyObstacles(pointCloudX,pointCloudY,pointCloudX,pointCloudY,diff,vehicleSpe
eds,beams);
        ArrayList Obstacle_Array = segmentation->getObstacleArray();
        // Update threat
        // Lock threat mutex (mutual exclusion)
        pthread_mutex_lock(&threat_mutex);
        int threat = segmentation->returnThreat(); // make it getThreat()
for convention

        pthread_mutex_unlock(&threat_mutex);

```

```

#ifndef DEBUG
    cout<<"Threat: "<<threat<<endl;
#endif
if(threat>0 || fullAuto) {
    gpio_set_value(LED4, 1);
    ArrayDbl Road = segmentation->returnRoadBounds();
    ArrayDbl Vehicle; Vehicle.push_back(0.0);
    Vehicle.push_back(0.0); Vehicle.push_back(Road[2]);
        Vehicle.push_back(avgspeed*1.466667);
        //Vehicle.push_back(10.0);

if(isfinite(Road[0])&&isfinite(Road[1])&&isfinite(Road[2])) {
    rrt->Run_RRT(Vehicle, Obstacle_Array, Road);
    // ArrayList Final_Path = rrt->getFinalPath();
    // float steerFloat; // steerangle in rads
    if(rrt->getRRTVerify()) {
        #ifdef DEBUG
            cout<<"GOOD PATH"<<endl;
        #endif
        float steerFloat = -rrt->getRRTSteer();
        // left steer decreases servo duty cycle while
right steer increases, even though left steer is defined as positive in coord sxes

pthread_mutex_lock(&motor_update_mutex);
steerInt = (unsigned char)
(steerFloat*7.5f+127.67f);

pthread_cond_signal(&motor_update_cond);
pthread_mutex_unlock(&motor_update_mutex);
validPath = 1;
gpio_set_value(LED3,0);
}
else {
    pthread_mutex_lock(&threat_mutex);
    threat = 0;
    pthread_mutex_unlock(&threat_mutex);
    validPath = 0;
    #ifdef DEBUG
        cout<<"BAD PATH"<<endl;
    #endif
    gpio_set_value(LED3,1);

    // Emergency Braking

}
#endif DEBUG

```

```

cout<<"Steer:
"<<steerFloat*(180.0/M_PI)<<endl;
#endif
}

else {
    gpio_set_value(LED4, 0);
}
delete[] pointCloudX;
delete[] pointCloudY;
}

else {
#ifndef DEBUG
    cout<<"Urg_driver::get_distance(): "<<urg->what()
<<endl;
#endif
pthread_mutex_lock(&motor_update_mutex);
steerInt = 127;
throttleInt = 128;
pthread_cond_signal(&motor_update_cond);
pthread_mutex_unlock(&motor_update_mutex);

urg->stop_measurement();
urg->reboot();
urg->close();

usleep(50000);

urg->open(information.device_or_ip_name(),
information.baudrate_or_port_number(),
information.connection_type());
urg->start_measurement(Urg_driver::Distance,
Urg_driver::Infinity_times, 0);
}

if(indicateFreq)
    gpio_set_value(LED2,0);

clock_gettime(CLOCK_REALTIME,&requestEnd);
diff = ( requestEnd.tv_sec - requestStart.tv_sec ) + ( requestEnd.tv_nsec -
requestStart.tv_nsec ) / 1E9;
long diff_us = diff*1000000L;
long sleep_us = ps - diff_us - 130L;
if(sleep_us > 0) {
    timespec_add_us(&requestEnd, sleep_us);
}

```

```

        clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&requestEnd, NULL);
    }
    clock_gettime(CLOCK_REALTIME,&requestEnd);
    diff = ( requestEnd.tv_sec - requestStart.tv_sec ) + ( requestEnd.tv_nsec -
requestStart.tv_nsec ) / 1E9;
    elaspedTime += diff;
#ifndef DEBUG
    cout<< "Avoid Collisions Elasped Time: "<< diff << endl;
#endif
}
if(logOption > 1)
    ptCloudFile.close();
pthread_mutex_destroy(&threat_mutex);
urg->stop_measurement();
urg->laser_off();
urg->close();
#ifndef DEBUG
    cout<<"Exiting collision avoidance thread"<< endl;
#endif
pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RealTime/Tasks/MonitorBluetooth/monitorbluetooth.cpp

```

#include "monitorbluetooth.h"
#include "globaldata.h"
#include <fcntl.h>
#include <termios.h>
#include <pthread.h>
#include <math.h>

char* modemdevice; // extern shared variable
speed_t baudrate = B115200; // extern shared variable
double poll_timeout = 0.5; // extern shared variable

int fd;
struct termios oldtio, newtio;

void openSerialPort(void) {
    /* Open modem device for reading and writing and not as controlling tty
       because we don't want to get killed if linenoise sends CTRL-C. */
    fd = open(modemdevice, O_RDWR | O_NOCTTY );
    if (fd < 0) {
        cout<<"Failed to open serial port"<< endl;
        perror(modemdevice);

```

```

        pthread_mutex_lock(&global_RUN_mutex);
        globalRUN = false;
        pthread_mutex_unlock(&global_RUN_mutex);
    }

bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */
/* BAUDRATE: Set bps rate. You could also use cfsetispeed and cfsetospeed.
CRTSCTS : output hardware flow control (only used if the cable has
all necessary lines. See sect. 7 of Serial-HOWTO)
CS8   : 8n1 (8bit,no parity,1 stopbit)
CLOCAL : local connection, no modem control
CREAD  : enable receiving characters */
newtio.c_cflag = baudrate | CRTSCTS | CS8 | CLOCAL | CREAD;
/* IGNPAR : ignore bytes with parity errors
otherwise make device raw (no other input processing) */
newtio.c_iflag = IGNPAR;
/* Raw output */
newtio.c_oflag = 0;
/* ICANON : enable canonical input
disable all echo functionality, and don't send signals to calling program */
// newtio.c_lflag = ICANON;
/* now clean the modem line and */
tcflush(fd, TCIFLUSH);
/* activate the settings for the port */
tcsetattr(fd,TCSANOW,&newtio);
/* terminal settings done, now handle input*/
}

void haltMonitorBluetooth(void) {
    pthread_mutex_lock(&global_RUN_mutex);
    globalRUN = false;
    pthread_mutex_unlock(&global_RUN_mutex);

    pthread_mutex_lock(&motor_update_mutex);
    steerInt = 127;
    throttleInt = 128;
    #ifdef DEBUG
        cout<<"NUETRAL SET ON STOP"<<endl;
    #endif
    pthread_cond_signal(&motor_update_cond);
    pthread_mutex_unlock(&motor_update_mutex);
}

void *monitor_bluetooth(void *period) {
    long ps = *((long *) period);

    openSerialPort();
}

```

```

struct timespec timeoutStart, timeoutEnd;

double diff = 0.0;
double mainTdiff = 0.0;

while(globalRUN) {
    clock_gettime(CLOCK_REALTIME,&timeoutStart);

    char buf[20];

    // if system is not threatened use Bluetooth user control commands/signals
    int res = read(fd, buf, 2);
    if(res >=2) {
        // check for changes
        if((steerInt != buf[0])||(throttleInt != buf[1])) {
            pthread_mutex_lock(&motor_update_mutex);
            if(threat<1) {
                if(fullAuto<1)
                    steerInt = buf[0];
                throttleInt = buf[1];
            } else {
                if(holdThrottle<1)
                    throttleInt = buf[1];
            }
            pthread_cond_signal(&motor_update_cond);
            pthread_mutex_unlock(&motor_update_mutex);
        }
        if((steerInt == 0)&&(throttleInt == 0)) {
            haltMonitorBluetooth();
        } else if((steerInt == 1)&&(throttleInt == 1)) {
            haltMonitorBluetooth();
            system("shutdown -h now");
        }
        diff = 0.0;
    } else {
        diff += mainTdiff;
        if(diff > poll_timeout){
            pthread_mutex_lock(&motor_update_mutex);
            steerInt = 127;
            throttleInt = 128;
            #ifdef DEBUG
                cout<<"NUETRAL SET ON TIMEOUT"<<endl;
            #endif
            pthread_cond_signal(&motor_update_cond);
            pthread_mutex_unlock(&motor_update_mutex);
        }
    }
}

```

```

        diff = 0.0;
    }
}

short tempOutConv = (short)round(avgspeed*100.0f);
unsigned char dataout[2];
dataout[0] = (unsigned char) tempOutConv;
dataout[1] = (unsigned char) (tempOutConv >> 8);
if(threat<1) {
    dataout[1] = (unsigned char)(dataout[1] & 0x7F);
} else {
    dataout[1] = (unsigned char)(dataout[1] | 0x80);
    if(validPath>0)
        dataout[1] = (unsigned char)(dataout[1] | 0x40);
    else
        dataout[1] = (unsigned char)(dataout[1] & 0xBF);
}

int written = write(fd, dataout, 2);

clock_gettime(CLOCK_REALTIME,&timeoutEnd);
mainTdiff = ( timeoutEnd.tv_sec - timeoutStart.tv_sec ) + (
timeoutEnd.tv_nsec - timeoutStart.tv_nsec ) / 1E9;
long diff_us = mainTdiff*1000000L;
long sleep_us = ps - diff_us - 165L;
if(sleep_us > 0) {
    timespec_add_us(&timeoutEnd, sleep_us);
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&timeoutEnd, NULL);
}
clock_gettime(CLOCK_REALTIME,&timeoutEnd);
mainTdiff = ( timeoutEnd.tv_sec - timeoutStart.tv_sec ) + (
timeoutEnd.tv_nsec - timeoutStart.tv_nsec ) / 1E9;
#endif DEBUG
cout<< "Monitor Bluetooth Elasped Time: "<< mainTdiff << endl;
#endif
}

pthread_mutex_destroy(&motor_update_mutex);
pthread_cond_destroy(&motor_update_cond);
close(fd);
#endif DEBUG
cout<<"Exiting monitor bluetooth thread"<<endl;
#endif
pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RealTime/Tasks/MonitorEncoders/monitorencoders.cpp

```
#include "monitorencoders.h"
#include <pthread.h>
#include "encoder.h"
#include "globaldata.h"
#include "constants.h"
volatile float avgsped;
pthread_mutex_t encoders_mutex;
/*****************/
/* Monitor Encoders Speed Task
*****************/
void *monitor_encoders(void *period) {
    // Declare variables on thread's stack memory
    long ps = *((long *) period);
    struct timespec requestStart, requestEnd, encoderStart;
    int chk;
    // Initialize the encoder mutex (mutual exclusion)
    chk = pthread_mutex_init(&encoders_mutex, NULL);
    if(chk) {
        perror("pthread_mutex create failed\n");
        pthread_exit(NULL);
    }
    // Declare encoders on this thread's heap memory
    Encoder *encoder1 = new Encoder(&FL_ENCODER_PIN);
    Encoder *encoder2 = new Encoder(&FR_ENCODER_PIN);
    Encoder *encoder3 = new Encoder(&RL_ENCODER_PIN);
    Encoder *encoder4 = new Encoder(&RR_ENCODER_PIN);
    // Start the encoder polling threads/tasks
    encoder1->start(encoder1);
    encoder2->start(encoder2);
    encoder3->start(encoder3);
    encoder4->start(encoder4);
    // Monitor Encoders Speed Loop
    clock_gettime(CLOCK_REALTIME,&encoderStart);
    while(globalRUN) {
        clock_gettime(CLOCK_REALTIME,&requestStart);

        // Procedure to update shared encoder data
        // For each encoder:
        // lock mutex (mutual exclusion)
        // get new end-time
        // calc diff in time
        // update encoder speed and reset encoder count
        // unlock mutex (mutual exclusion)
        // no condition mutex to allow for zero velocity detection
```

```

double diff;

if(pthread_mutex_trylock( &(encoder1->encoder_mutex) ) == 0) {
    clock_gettime(CLOCK_REALTIME,&requestEnd);
    diff = ( requestEnd.tv_sec - encoderStart.tv_sec ) + (
requestEnd.tv_nsec - encoderStart.tv_nsec ) / 1E9;
    encoder1->setSpeed( encoder1-
>getCount()*encoderStepsize*(1.0/5280.0)*(1.0/diff)*3600.0 );
    encoder1->resetCount();
    pthread_mutex_unlock( &(encoder1->encoder_mutex) );
}

if(pthread_mutex_trylock( &(encoder2->encoder_mutex) ) == 0) {
    clock_gettime(CLOCK_REALTIME,&requestEnd);
    diff = ( requestEnd.tv_sec - encoderStart.tv_sec ) + (
requestEnd.tv_nsec - encoderStart.tv_nsec ) / 1E9;
    encoder2->setSpeed( encoder2-
>getCount()*encoderStepsize*(1.0/5280.0)*(1.0/diff)*3600.0 );
    encoder2->resetCount();
    pthread_mutex_unlock( &(encoder2->encoder_mutex) );
}

if(pthread_mutex_trylock( &(encoder3->encoder_mutex) ) == 0) {
    clock_gettime(CLOCK_REALTIME,&requestEnd);
    diff = ( requestEnd.tv_sec - encoderStart.tv_sec ) + (
requestEnd.tv_nsec - encoderStart.tv_nsec ) / 1E9;
    encoder3->setSpeed( encoder3-
>getCount()*encoderStepsize*(1.0/5280.0)*(1.0/diff)*3600.0 );
    encoder3->resetCount();
    pthread_mutex_unlock( &(encoder3->encoder_mutex) );
}

if(pthread_mutex_trylock( &(encoder4->encoder_mutex) ) == 0) {
    clock_gettime(CLOCK_REALTIME,&requestEnd);
    diff = ( requestEnd.tv_sec - encoderStart.tv_sec ) + (
requestEnd.tv_nsec - encoderStart.tv_nsec ) / 1E9;
    encoder4->setSpeed( encoder4-
>getCount()*encoderStepsize*(1.0/5280.0)*(1.0/diff)*3600.0 );
    encoder4->resetCount();
    pthread_mutex_unlock( &(encoder4->encoder_mutex) );
}

/// \todo check if all encoders values are close - otherwise return error on
log file at pin # and tire of faulty encoder
pthread_mutex_lock(&encoders_mutex);

```

```

    avgsspeed = (encoder1->getSpeed() + encoder2->getSpeed() + encoder3-
>getSpeed() + encoder4->getSpeed()) / 4.0f;
    cout << "Average speed: " << avgsspeed << endl;
    pthread_mutex_unlock(&encoders_mutex);
    clock_gettime(CLOCK_REALTIME, &requestEnd);

    clock_gettime(CLOCK_REALTIME, &encoderStart);
    diff = (requestEnd.tv_sec - requestStart.tv_sec) + (requestEnd.tv_nsec -
requestStart.tv_nsec) / 1E9;
    long diff_us = diff * 1000000L;
    long sleep_us = ps - diff_us - 165L;
    if(sleep_us > 0) {
        timespec_add_us(&requestEnd, sleep_us);
        clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&requestEnd, NULL);
    }
    clock_gettime(CLOCK_REALTIME, &requestEnd);
    diff = (requestEnd.tv_sec - requestStart.tv_sec) + (requestEnd.tv_nsec -
requestStart.tv_nsec) / 1E9;
    cout << "Monitor Encoders Elapsed Time: " << diff << endl;
}
pthread_mutex_destroy(&encoders_mutex);
int rc = pthread_join(encoder1->getMonitorThread(), NULL);
rc = pthread_join(encoder2->getMonitorThread(), NULL);
rc = pthread_join(encoder3->getMonitorThread(), NULL);
rc = pthread_join(encoder4->getMonitorThread(), NULL);
delete encoder1;
delete encoder2;
delete encoder3;
delete encoder4;
cout << "Exiting monitor encoders thread" << endl;
pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RealTime/Tasks/MonitorEncoder/encoder.cpp

```

#include "encoder.h"
#include "globaldata.h"

Encoder::Encoder(const void *pNum) {
    (this->encoder_thread_data).pinNum = *((int *) pNum);
    cout << "Encoder pin is: " << (this->encoder_thread_data).pinNum << endl;
}
void Encoder::start(Encoder *encoder_p) {
    pthread_attr_t my_attr;
    struct sched_param param;

```

```

pthread_attr_init(&my_attr);
pthread_attr_setinheritsched(&my_attr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&my_attr, SCHED_RR );

param.sched_priority = 1;
pthread_attr_setschedparam(&my_attr, &param);

cout<<"Encoder Pin is: "<<((encoder_p->encoder_thread_data).pinNum)<<endl;
int chk = pthread_create( &((encoder_p->encoder_thread_data).monitor_thread),
&my_attr, encoder_p->monitor_wrapper, encoder_p );
if(chk) { cout << "unable to create encoder thread @ pin " << ((encoder_p-
>encoder_thread_data).pinNum) << endl; }

pthread_attr_destroy(&my_attr);
}

/***********************/

* Encoder Polling Task (ideally implemented as an interrupt)
/***********************/

void *Encoder::monitor_encoder(void *context) {
//cout<<"From new thread"<<endl;
struct pollfd fdset[2];
int nfds = 2;
int gpio_fd, timeout, rc;
char *buf[MAX_BUF];
unsigned int gpio;
int len;

gpio = (((Encoder *)context)->encoder_thread_data).pinNum;

int chk;
chk = pthread_mutex_init( &((Encoder *)context)->encoder_mutex), NULL);
if(chk) {
    perror("pthread_mutex create failed\n");
    pthread_exit(NULL);
}

gpio_export(gpio);
gpio_set_dir(gpio, 0);
gpio_set_edge(gpio, "rising");
gpio_fd = gpio_fd_open(gpio);

if(gpio_fd < 0) {
    perror("Failed\n");
    pthread_mutex_lock(&global_RUN_mutex);
    globalRUN = false;
}

```

```

        pthread_mutex_unlock(&global_RUN_mutex);
    }

timeout = GPIO_POLL_TIMEOUT;

struct timespec next, start;

while (globalRUN) {
    clock_gettime(CLOCK_REALTIME, &start);

    memset((void*)fdset, 0, sizeof(fdset));

    fdset[0].fd = STDIN_FILENO;
    fdset[0].events = POLLIN;

    fdset[1].fd = gpio_fd;
    fdset[1].events = POLLPRI;

    rc = poll(fdset, nfds, timeout);

    if (rc < 0) {
        printf("\nFailed to poll encoder @ pin# \n");
    }

    if (fdset[1].revents & POLLPRI) {
        read(fdset[1].fd, buf, MAX_BUF);
        int chk = pthread_mutex_trylock(&(((Encoder *)context)->encoder_mutex));
        if(chk == 0) {
            (((Encoder *)context)->encoder_thread_data).encoder_count++;
            pthread_mutex_unlock(&(((Encoder *)context)->encoder_mutex));
        } else {
            if(chk == EBUSY)
                cout<<"Thread Busy"<<endl;
            else if(chk == EINVAL)
                cout<<"Protocol Attr > Mutex Priority
Ceiling"<<endl;
            else if(chk == EAGAIN)
                cout<<"Max num of recursive locks for mutex
exceeded"<<endl;
        }
    }

    if (fdset[0].revents & POLLIN) {

```

```

        (void)read(fdset[0].fd, buf, 1);
    }

fflush(stdout);

clock_gettime(CLOCK_REALTIME, &next);
double diff = ( next.tv_sec - start.tv_sec ) + ( next.tv_nsec - start.tv_nsec ) / 1E9;
long diff_us = diff*1000000L;
long sleep_us = 1860L - diff_us - 165L;

if(sleep_us > 0) {
    timespec_add_us(&next, sleep_us); // sample encoder value @
3585 Hz = 280 us
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&next, NULL);
}
clock_gettime(CLOCK_REALTIME, &next);
diff = ( next.tv_sec - start.tv_sec ) + ( next.tv_nsec - start.tv_nsec ) / 1E9;
cout<< "Monitor Encoder " << gpio << " Elapsed Time: "<< diff << endl;
}

pthread_mutex_destroy( &(((Encoder *)context)->encoder_mutex) );
// close gpio
gpio_fd_close(gpio_fd);
cout<<"Exiting encoder monitor thread for pin: "<<gpio<<endl;
pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RealTime/Tasks/MotorUpdater/motorupdater.cpp

```

#include "motorupdater.h"
#include <pthread.h>
#include "globaldata.h"
#include "constants.h"
volatile int throttleInt;
volatile int steerInt;
pthread_mutex_t motor_update_mutex;
pthread_cond_t motor_update_cond;
PWM::Pin *pinA;
PWM::Pin *pinB;

void cleanMotorUpdater(void *arg) {
    pinA->SetDutyPercent( STEER_NEUTRAL );
    pinB->SetDutyPercent( THROTTLE_NEUTRAL );
    pinA->Disable();
    pinB->Disable();
}

```

```

        delete pinA;
        delete pinB;
        pthread_mutex_destroy(&motor_update_mutex);
        pthread_cond_destroy(&motor_update_cond);
        cout<<"Exiting motor updater thread"<<endl;
    }
/************************************************************************/
* Motor Updater Task
/************************************************************************/
void *motor_updater(void *period) {
    const long periodNS = *((long *)period); // PWM period - no period as task is
    triggered to run when shared data changes
    // Initialize PWM pin for
    pinA = new PWM::Pin(STEERPIN, periodNS);
    // Initialize PWM pin for
    pinB = new PWM::Pin(THROTTLEPIN, periodNS);

    // Load PWM pin drivers
    PWM::GetCapeManagerSlot(STEERPIN);
    PWM::GetCapeManagerSlot(THROTTLEPIN);

    // PWM neutral state steer and throttle
    pinA->SetDutyPercent( STEER_NEUTRAL );
    pinB->SetDutyPercent( THROTTLE_NEUTRAL );

    pinA->Enable();
    pinB->Enable();

    int chk;
    chk = pthread_mutex_init(&motor_update_mutex, NULL);
    if(chk) {
        perror("pthread_mutex create failed\n");
        pthread_exit(NULL);
    }
    chk = pthread_cond_init(&motor_update_cond,NULL);
    if(chk) {
        perror("pthread_mutex cond create failed\n");
        pthread_exit(NULL);
    }

    pthread_cleanup_push(cleanMotorUpdater, NULL);

    while(globalRUN) {
        // Run as task to update only when values change by using a mutex
        (mutual exclusion) and condition signal
        pthread_mutex_lock(&motor_update_mutex);

```

```

        // Wait for thread to signal that motor values have changed
        pthread_cond_wait(&motor_update_cond,&motor_update_mutex);
        float steerDuty = 0.0999f+(float)(steerInt*0.0003f);
        float throttleDuty = 0.0999f+(float)(throttleInt*0.0004f);
        // cout<<"DUTY CYCLE UPDATED"<<endl;
        pinA->SetDutyPercent( steerDuty );
        pinB->SetDutyPercent( throttleDuty );
        pthread_mutex_unlock(&motor_update_mutex);
    }

    pthread_cleanup_pop(0);
    pinA->SetDutyPercent( STEER_NEUTRAL );
    pinB->SetDutyPercent( THROTTLE_NEUTRAL );
    pinA->Disable();
    pinB->Disable();
    delete pinA;
    delete pinB;
    pthread_mutex_destroy(&motor_update_mutex);
    pthread_cond_destroy(&motor_update_cond);
    cout<<"Exiting motor updater thread"<<endl;
    pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RealTime/Tasks/LogData/logdata.cpp

```

#include "logdata.h"
#include "globaldata.h"
#include <fstream>
#include <ctime>

std::queue<std::string> logdataqueue;
pthread_mutex_t log_data_mutex;
std::string logfilepath;

char timeString[20];

void write_text_to_log_file( const std::string &text ) {
    std::ofstream log_file( logfilepath.c_str(), std::ios_base::out | std::ios_base::app );
    log_file << text << std::endl; // std::endl flushes and stream closes when function exits
}

void createLogFile(void) {
    char timeString[20];
    time_t current_time = time(NULL);

    // mount SD card
    // std::string basePath ("~/media/card/datalog_");

```

```

        std::string baseFilePath ("/media/datalog_");
        strftime(timeString, sizeof(timeString), "%Y-%m-%d_%H-%M-%S",
localtime(&current_time));
        std::string time_String(timeString);

        logfilepath = baseFilePath+time_String;
#ifndef DEBUG
        cout<<"Logging data to "<<logfilepath<<endl;
#endif
    }
void *log_data(void *period) {
    long ps = *((long *) period);
    struct timespec requestStart, requestEnd, logStart;
    int chk;
    double elaspedTime = 0.0;
    // Initialze the mutex (mutual exclusion)
    chk = pthread_mutex_init(&log_data_mutex, NULL);
    if(chk) {
        perror("pthread_mutex create failed\n");
        pthread_exit(NULL);
    }

    // create log file path string with unique date-timestamp
createLogFile();

    time_t current_time = time(NULL);
    strftime(timeString, sizeof(timeString), "%H:%M:%S",
localtime(&current_time));
    std::string time_String(timeString);
    write_text_to_log_file(time_String);

    clock_gettime(CLOCK_REALTIME,&logStart);
    while(globalRUN) {
        clock_gettime(CLOCK_REALTIME,&requestStart);
        pthread_mutex_lock(&log_data_mutex);
        ostringstream convert;
        convert << elaspedTime << " ";
        convert << throttleInt << " ";
        convert << steerInt << " ";
        convert << avgSpeed << " ";
        convert << threat; // << " "; // Indicates path-planning algorithm was run
        //convert << validPath; // Indicates if path was
successfully determined (if threat and validPath are 1 then steerInt corresponds to
autonomous steer)
        write_text_to_log_file(convert.str());
        // if queue is not empty - write text to log file
    }
}

```

```

        //while(!logdataqueue.empty())
        //    write_text_to_log_file(logdataqueue.pop());
        //}
        pthread_mutex_unlock(&log_data_mutex);
        clock_gettime(CLOCK_REALTIME,&requestEnd);
        double diff = ( requestEnd.tv_sec - requestStart.tv_sec ) + (
requestEnd.tv_nsec - requestStart.tv_nsec ) / 1E9;
        long diff_us = diff*1000000L;
        long sleep_us = ps - diff_us - 165L;
        if(sleep_us > 0) {
            timespec_add_us(&requestEnd, sleep_us);
            clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME,
&requestEnd, NULL);
        }
        clock_gettime(CLOCK_REALTIME,&requestEnd);
        diff = ( requestEnd.tv_sec - requestStart.tv_sec ) + ( requestEnd.tv_nsec -
requestStart.tv_nsec ) / 1E9;
        elaspedTime += diff;
        #ifdef DEBUG
            cout<< "Log Data Elasped Time: "<< diff << endl;
        #endif
    }
    pthread_mutex_destroy(&log_data_mutex);
    // Compress recorded data file
    std::string systemCall = "tar -cvzf "+filepath+".tar.gz "+filepath;
    int status = system(systemCall.c_str());
    #ifdef DEBUG
        cout<<"Exiting log data thread"<<endl;
    #endif
    pthread_exit(NULL);
}

```

root/src/BeagleBone/source/RRT/RRT\_class.cpp

```

/*
 * RRT_class.cpp
 *
 * Created on: Apr 16, 2014
 * Author: Thomas
 */
//RRT_class.cpp
#include "RRT_class.h"
#include <bits/random.h>
random_device rd;
default_random_engine generator;
//mt19937 re(rd());

```

```

mt19937 re(time(0));
uniform_real_distribution<double> ud(0.0,1.0);
RRT::ABA::ABA(double targetFT, _sensor_type st = LIDAR) {
    target = targetFT;
    ABA_event = false;
    brake_percent = 0;
    prev_distABA = 0;
    curr_distABA = 0;
    Vrel = 0;
    TTC = 0;
    updateFirstDistFlag = true;
    sensor_type = st;
    GAIN = 1.25;
}
void RRT::ABA::zero() {
    ABA_event = false;
    brake_percent = 0;
}
// LiDAR based ABA update
void RRT::ABA::update(double speed, ArrayList Obstacle_Array, double deltaT) {
    if(Obstacle_Array.size() > 0) {
        updateFirstDistFlag = true;
        for(int i=0; i < Obstacle_Array.size(); i++) {
            ArrayDbl obstacle = Obstacle_Array[i];
            if(((obstacle[0]+obstacle[2]/2) < -0.5)||((obstacle[0]-obstacle[2]/2)
> 0.5))
                ;
            else {
                if(((obstacle[1]-obstacle[3]/2) < curr_distABA) ||
updateFirstDistFlag) {
                    curr_distABA = (obstacle[1]-obstacle[3]/2);
                    updateFirstDistFlag = false;
                }
            }
        }
        if(prev_distABA!=0) {
            Vrel = (prev_distABA - curr_distABA)/deltaT;
        } else {
            Vrel = 0;
        }
        TTC = curr_distABA/(speed*3.2808399-Vrel);
        if(TTC < target && TTC > 0) {
            ABA_event = true;
            brake_percent = (int)((target-TTC)/target)*100*GAIN;
            if(brake_percent > 100)
                brake_percent = 100;
        }
    }
}

```

```

        } else {
            brake_percent = 0;
            ABA_event = false;
        }
        cout<<endl<<"ABA Event: "<<ABA_event<<endl<<"CurrDist:
"<<curr_distABA<<endl<<"PrevDist: "<<prev_distABA<<endl<<"TTC:
"<<TTC<<endl<<"Vrel: "<<Vrel<<endl<<"Brake %: "<<brake_percent<<endl;

        prev_distABA = curr_distABA;
    }
}

// Ultrasonic based ABA update
void RRT::ABA::update(double speed, double distance, double deltaT) {
    if(distance > 0) {

        curr_distABA = distance;

        if(prev_distABA!=0) {
            Vrel = (prev_distABA - curr_distABA)/deltaT;
        } else {
            Vrel = 0;
        }
        TTC = curr_distABA/(speed*3.2808399-Vrel);
        if(TTC < target && TTC > 0) {
            ABA_event = true;
            brake_percent = (int)((target-TTC)/target)*100;
        } else {
            brake_percent = 0;
            ABA_event = false;
        }
        cout<<endl<<"ABA Event: "<<ABA_event<<endl<<"CurrDist:
"<<curr_distABA<<endl<<"PrevDist: "<<prev_distABA<<endl<<"TTC:
"<<TTC<<endl<<"Vrel: "<<Vrel<<endl<<"Brake %: "<<brake_percent<<endl;

        prev_distABA = curr_distABA;
    }
}

RRT::RRT() {
    minTimeToCollision = 3.5;
    enableABA = true;
    if(enableABA)
        aba = new ABA(minTimeToCollision);
}

RRT::~RRT() {
    delete aba;
}

```

```

void RRT::Run_RRT(ArrayDbl Vehicle,ArrayList Obstacle_Array,ArrayDbl Road) {

    memset( Road_Lines,0,sizeof(Road_Lines) );
    memset( Temp_Node,0,sizeof(Temp_Node) );
    ArrayList Tree;
    memset( Route_Tree,0,sizeof(Route_Tree) );
    double Final_Tree[20][2];
    memset( Final_Tree,0,sizeof(Final_Tree) );
    float Node_Dist = 2.5f; // distance between nodes in tree
    bool Done = false;
    int Nodes = 1;
    bool Rand_Go = false;
    bool Fail = false;
    int Iterations = 0;
    float Vehicle_Geo_Actual[] = { 1.75f, 0.833f };
    float Buffer = 0.15f;
    float Vehicle_Geo[] = { Vehicle_Geo_Actual[0]+Buffer,
Vehicle_Geo_Actual[1]+Buffer };
    memset( Final_Path, 0, sizeof(Final_Path) );

    aba->zero();

    if(Vehicle[3] == 0) {
        Vehicle[3] = 0.0000001;
    }

    failedPaths.clear();
    //Num_Objects
    int ObstacleNum = Obstacle_Array.size();

    // No Road Detected
    if( !roadDetected( Road ) )
        Fail = true;
    // Road aligned along car axis
    if(Road[2] == 0)
        Road[2] = 0.0000001;
    // GET LINES OF ROAD BOUNDARIES
    Get_Roadlines( Road );
    // GET GOAL
    Get_Goal( Road );
    srand(time(NULL));
    while((Done == false) && (Fail == false)) {

        bool NoGoGo = false;
        // ADD A TEMPORARY NODE TO [TREE]
        if(Tree.size() == 0) {

```

```

        ArrayDbl tempPush;
        tempPush.push_back( 0 ); tempPush.push_back( 0 );
tempPush.push_back( 0 );
        Tree.push_back( tempPush ); }
if(Rand_Go == false) {
        AddLinear(Node_Dist, Tree, Nodes);
}
else {
        AddRandom(Node_Dist, Tree, Nodes, Road);
}
// CHECK TO SEE IF FINISHED
if((Temp_Node[0] == Goal[0])&&(Temp_Node[1] == Goal[1]))
        Done = true;
// GENERATE A PATH TO THE TEMPORARY NODE
Get_Path(Tree, Node_Dist, Vehicle, Vehicle_Geo );
// CHECK IF IT WAS A GOOD PATH
if(Path[0][0] > 9000) {
        NoGoGo = true;
}
else {
        // PATH IS GOOD, NOW CHECK FOR COLLISION
        // MULTIPLE OBJECT LOOP
        for(int i=0; i < ObstacleNum; i++) {
                ArrayDbl Obstacle = Obstacle_Array[i];
                NoGoGo = CheckCollision(Obstacle, Vehicle_Geo, Road);
                if(NoGoGo)
                        break;
        }
}
// ADD TEMPORARY NODE TO THE TREE IF IT IS VALID
if(NoGoGo == false) {
        Nodes = Nodes + 1;                                // INCREMENT THE
NODE COUNTER
        ArrayDbl tempAdd; tempAdd.push_back( Temp_Node[0] );
tempAdd.push_back( Temp_Node[1] ); tempAdd.push_back( Temp_Node[2] );
        Tree.push_back(tempAdd);                         // ADDS THE
NODE TO [TREE]
        Rand_Go = false;                               // STOPS CHECKING
RANDOM NODES
        } else {
// TRY A NEW RANDOM NODE
        Rand_Go = true;
#ifndef STORE_FAILED_PATHS
        vector<double> steerandtimeValues;
        vector< vector<double> > retsteerandtimeValues;
        for(int i = 0; i < 50; i++) {

```

```

                steerandtimeValues.push_back(Path[i][0]);
                steerandtimeValues.push_back(Path[i][1]);
                steerandtimeValues.push_back(Path[i][2]);
                steerandtimeValues.push_back(Path[i][3]); // push
back steer
                steerandtimeValues.push_back(Path[i][4]); // push
back time

        retsteerandtimeValues.push_back(steerandtimeValues);
                steerandtimeValues.clear();
    }
    failedPaths.push_back(retsteerandtimeValues);
    retsteerandtimeValues.clear();
#endif
}
if(Done) {
    copy(&Path[0][0], &Path[0][0]+50*5,&Final_Path[0][0]);
    int connectingNode = Nodes - 1; // INITIALIZES THE CONNECTION NODE FROM THE GOAL
    int finalNodes = 0;

    while(connectingNode >= 0) {
        ArrayDbl tempSet = Tree[connectingNode];
        Final_Tree[finalNodes][0] = tempSet[0];
        Final_Tree[finalNodes][1] = tempSet[1]; // MOVES THE PREVIOUS NODE IN THE
        TREE INTO THE FINAL TREE
        connectingNode = tempSet[2] - 1; // SETS THE NEXT CONNECTING NODE
        finalNodes = finalNodes + 1;
    }
    Iterations = Iterations + 1;
    if(Iterations > 200) {
        Fail = true;
    }
}
if(Fail) {
    Final_Path[0][0] = 9001;
    if(enableABA)
        aba->update(Vehicle[3], Obstacle_Array, 0.05);
}

// for(unsigned int i = 0; i < Tree.size(); i++) {
//     ArrayDbl tempPrint = Tree[i];
//     for( int j = 0; j < 3; j++) {
//         cout<<tempPrint[j]<<" ";

```

```

// }
// cout<<endl;
// }
// cout<<"Final Path"<<endl;
// for(int i = 0; i < 50; i++) {
    // for(int j = 0; j < 5; j++) {
        // cout<<Final_Path[i][j]<< " ";
    // }
    // cout<<endl;
}
vector< vector< vector<double> > > RRT::getFailedPaths() {
    return failedPaths;
}

bool RRT::roadDetected(ArrayDbl Road) {
    for(unsigned int i=0; i < Road.size();i++ ) {
        if(Road[i]!=0)
            return true;
    }
    return false;
}
float RRT::getRRTSteer() {
    return Final_Path[0][3];
}
bool RRT::getRRTVerify() {
    if(Final_Path[0][0] > 9000)
        return 0;
    else
        return 1;
}
vector< vector<double> > RRT::getFinalPath() {
    vector<double> steerandtimeValues;
    vector< vector<double> > retsteerandtimeValues;
    if(getRRTVerify()) { //&& not empty
        for(int i = 0; i < 50; i++) {
            steerandtimeValues.push_back(Final_Path[i][0]); // push back x-
position
            steerandtimeValues.push_back(Final_Path[i][1]); // push back y-
position
            steerandtimeValues.push_back(Final_Path[i][2]); // push back yaw
angle
            steerandtimeValues.push_back(Final_Path[i][3]); // push back
steer
            steerandtimeValues.push_back(Final_Path[i][4]); // push back time
    }
}

```

```

        retsteerandtimeValues.push_back(steerandtimeValues);
        steerandtimeValues.clear();
    }

    memcpy(Prev_Path, Final_Path, sizeof(Prev_Path));
}

else {
    // for(int i = 0; i < 50; i++) {
        // steerandtimeValues.push_back(Path[i][0]);
        // steerandtimeValues.push_back(Path[i][1]);
        // steerandtimeValues.push_back(Path[i][2]);
        // steerandtimeValues.push_back(Path[i][3]); // push back steer
        // steerandtimeValues.push_back(Path[i][4]); // push back time
        // retsteerandtimeValues.push_back(steerandtimeValues);
        // steerandtimeValues.clear();
    // }

    // failedPaths.push_back(retsteerandtimeValues);
    // retsteerandtimeValues.clear();

    // for(int i = 0; i < 50; i++) {
        // steerandtimeValues.push_back(Prev_Path[i][0]);
        // steerandtimeValues.push_back(Prev_Path[i][1]);
        // steerandtimeValues.push_back(Prev_Path[i][2]);
        // steerandtimeValues.push_back(Prev_Path[i][3]); // push back
steer
time
        // steerandtimeValues.push_back(Prev_Path[i][4]); // push back
        // retsteerandtimeValues.push_back(steerandtimeValues);
        // retsteerandtimeValues.clear();
        // steerandtimeValues.clear();
    // }

    return retsteerandtimeValues;
}

void RRT::Get_Roadlines(ArrayDbl Road) {
    double xl = Road[0];           // perpendicular distance from lidar to left road
boundary
    double xr = Road[1];           // perpendicular distance from lidar to right road
boundary
    double yaw = Road[2];          // angle between y-lidar coordinate and road
[degrees]
    // yaw is positive in the counterclockwise direction

    // RIGHT ROAD EQUATION y= m *x + br
}

```

```

        double br = (-xr)/tan( yaw*(M_PI/180.0) ); // y-intercept
        double m = -br/xr; // slope

        // LEFT ROAD EQUATION y= m *x + bl
        double bl = (-xl)/tan( yaw*(M_PI/180.0) ); // y-intercept

        // ROADLINES=[m, bl, br] ;
        Road_Lines[0] = m;
        Road_Lines[1] = bl;
        Road_Lines[2] = br;
    }

void RRT::Get_Goal(ArrayDbl Road) {
    double xl = Road[0]; // perpendicular distance from lidar to left road
boundary
    double xr = Road[1]; // perpendicular distance from lidar to right road
boundary
    double yaw = Road[2]; // angle between y-lidar coordinate and road
[degrees]

    double m = Road_Lines[0];
    double Goal_Dist = 20.0; // distance down the road the goal is
from the car
    double xrr= abs( cos( yaw*(M_PI/180.0) )*xr ); // closest distance to road
from car on right
    double xlr= abs( cos( yaw*(M_PI/180.0) )*xl ); // closest distance to road
from car on left
    int SIGN = yaw/abs(yaw);

    double Y;
    double x_dist;
    double w;

    // car is aligned straight with road
    if (yaw == 0) {
        Goal[0] = (xr+xl)/2;
        Goal[1] = Goal_Dist;
    }
    else {
        // car xl is the same as xr
        if (abs(xl)== xr)
            Y = cos( yaw*(M_PI/180.0) )*Goal_Dist;
        // car on the left side of road
        if (abs(xl) < xr) {
            x_dist = xrr - (xlr + xrr)/2;
            w = sin( yaw*( M_PI/180.0 ) ) * x_dist;
        }
    }
}

```

```

        Y = cos( yaw*(M_PI/180.0) ) * Goal_Dist + SIGN*w;
    }
    // car is on the right side of road
    if (abs(xl)> xr) {
        x_dist = xlr - (xlr + xrr)/2;
        w = sin( abs(yaw)*(M_PI/180.0) )*x_dist;
        Y = cos( yaw*(M_PI/180.0) ) * Goal_Dist + SIGN*w;
    }
    // now using a centerline of road to find x value
    // if it is on the right side of road middle line will have - yint
    double Bl = Road_Lines[1]; // y-intercept of left road line
    double Br = Road_Lines[2]; // y-intercept of right road line
    double Bc = (Bl + Br)/2; // y-intercept of center road line

    // road center line
    double X = (Y - Bc)/m; // X value of goal
    Goal[0] = X;
    Goal[1] = Y;
}
}

void RRT::AddLinear( float Node_Dist , ArrayList Tree , int Nodes ) {
    ArrayDbl Close_Node = Tree[Nodes-1]; // SETS NODE JUST ADDED TO THE
    CLOSEST NODE
    // TO TEST A STRAIGHT PATH FROM IT

    // DETERMINES THE DISTANCE TO THE GOAL
    double Goal_Dist = pow( pow( Goal[0] - Close_Node[0], 2 ) + pow( Goal[1] -
    Close_Node[1], 2 ), 0.5 );
    // CHECK TO SEE IF THE GOAL HAS BEEN REACHED BY COMPARING
    // THE DISTANCE TO THE GOAL TO THE NODE DISTANCE

    if(Goal_Dist <= Node_Dist) {
        Temp_Node[0] = Goal[0];
        Temp_Node[1] = Goal[1];
    }
    else {
        // LOCATION OF TEMPORARY NODE IN THE DIRECTION OF THE
        GOAL
        Temp_Node[0] = Close_Node[0] + (Node_Dist*(Goal[0] -
        Close_Node[0])) / Goal_Dist; // X
        Temp_Node[1] = Close_Node[1] + (Node_Dist*(Goal[1] -
        Close_Node[1])) / Goal_Dist; // Y
    }
    Temp_Node[2] = Nodes;
}

```

```

void RRT::AddRandom( float Node_Dist , ArrayList Tree, int Nodes , ArrayDbl Road ) {

    // CREATE RANDOM NODE ON ROAD
    // yaw is positive in the counterclockwise direction

    double xl = Road[0];                      // perpendicular distance from lidar to left
    road boundary
    double xr = Road[1];                      // perpendicular distance from lidar to right
    road boundary
    double yaw = Road[2];                      // angle between y-lidar coordinate
    and road
    double m = Road_Lines[0];                  // slope of road

    double Ymin = 2.0; // (xr-xl)/2;           // minimum distance in y
    direction to search in
    double Ymax = 30.0;                       // maximum distance in y direction to search
    in

    //double X = (xr-xl)*unifRandNumber() + xl;      // RANDOM x-point along
    line perpendicular to y-axis
    //double randTemp = ud(generator);
    double X = (xr-xl)*ud(re) + xl;
    //double Y = (Ymax-Ymin)*unifRandNumber() + Ymin; // RANDOM y-point
    along y-lidar axis
    double Y = (Ymax-Ymin)*ud(re) + Ymin;
    double Rand_Node[3];
    double Close_Node[2];
    double B_rand;

    if(yaw == 0) {                                // the road is parallel to y-lidar axis
        Rand_Node[0] = X;
        Rand_Node[1] = Y;
    }
    else {                                         // projects point along random line parallel
        and within the road
        // Solving y = mx+b for y = 0 and x = Xrandom to find b
        B_rand = -m*X;                            // random y-intercept based on X
        Rand_Node[0] = (Y - B_rand)/m; // RAND_NODE x value is then calculated
        based on Y
        Rand_Node[1] = Y;                          // RAND_NODE Y value is same as Y
    }

    // SOLVING FOR THE DISTANCE TO THE RANDOM NODE
    // INITIALIZES RAND_DIST TO A MAXIMUM VALUE WHICH WOULD
    BE

```

```

// THE DISTANCE TO THE START OF THE TREE
ArrayDbl temp = Tree[0];
double Rand_Dist = pow(pow( Rand_Node[0] - temp[0], 2 ) + pow(
Rand_Node[1] - temp[1], 2 ), 0.5 );

// DETERMINING CLOSEST NODE TO THE RANDOM NODE
for(int i = 0; i < Nodes; i++) {

    // SOLVING FOR THE DISTANCE TO THE RANDOM NODE FROM
    // THE A NODE IN [TREE]
    temp = Tree[i];
    double Temp_Dist = pow(pow( Rand_Node[0] - temp[0], 2 ) + pow(
Rand_Node[1] - temp[1], 2 ), 0.5 );

    if (Temp_Dist <= Rand_Dist) {
        Rand_Dist = Temp_Dist;           // MAKES THE CHECKED
        DISTANCE THE NEW DISTANCE
        Close_Node[0] = temp[0];         // SETS THE CLOSEST NODE
        Close_Node[1] = temp[1];
        Temp_Node[2] = i + 1;           // MARKS WHAT NODE IT
        IS
    }
}

// LOCATION OF TEMPORARY NODE
Temp_Node[0] = Close_Node[0] + (Node_Dist*(Rand_Node[0] -
Close_Node[0]))/Rand_Dist; // X
Temp_Node[1] = Close_Node[1] + (Node_Dist*(Rand_Node[1] -
Close_Node[1]))/Rand_Dist; // Y

if (Temp_Node[1] < Close_Node[1]) {
    temp = Tree[ Temp_Node[2]-1 ];
    Temp_Node[2] = temp[2];
}
}

double RRT::unifRandNumber() {
    // Seed Random Number Generator
    // srand(time(0));
    double randomNum = ((double) rand() / ((double)RAND_MAX));
    return randomNum;
}

void RRT::Get_Path(ArrayList Tree,float Node_Dist,ArrayDbl Vehicle,float
Vehicle_Geo[]) {

    float angleLimit = 17.0;           // MAXIMUM ALLOWABLE ANGLE OF
    THE FRONT WHEEL
}

```

```

        int Route_Count = 0;           // NUMBER OF NODES IN THE ROUTE TO BE
TESTED
        double center[] = { 0,0 };     // CENTER OF THE RADIUS OF CURVATURE
        double theta = 0;             // ANGLE OF THE CAR
        double phi = 0;               // ARC ANGLE
        double nu = 0;                // LOOKAHEAD ANGLE
        double LFW = 0;                // LOOKAHEAD DISTANCE
        double radius = 0;             // RADIUS OF CURVATURE
        int divLook = 8;                // NUMBER OF DIVISIONS TO

LOOKAHEAD
        double DIV = 4;                  // NUMBER OF DIVISIONS PER
TREE SEGMENT
        double angleDiff = 4;           // Differential step size along the arc
        int tests = 0;                  // NUMBER OF TESTS REQUIRED
        bool nogogo = false;            // FLAG TO DETERMINE PATH FEASIBILITY
        float length = Vehicle_Geo[0];   // LENGTH OF THE VEHICLE
        double timeDiff = 0;              // TIME BETWEEN NODES [s]
        double timeTot = 0;                // TOTAL TIME FROM THE ORIGIN [s]

pathCount = 0;
memset( Path,0,sizeof(Path) );
// FRAMES

// WORK THE ROUTE BACKWARDS

// EXTRACT THE COMPLETED
PATH*****
int connectingNode = (int)Temp_Node[2]; // INITIALIZES THE
CONNECTION NODE FROM THE TEMP NODE
Route_Tree[0][0] = Temp_Node[0];
Route_Tree[0][1] = Temp_Node[1];
Route_Count = 0;

// Divide each branch of the tree into sub-steps
while(connectingNode > 0) {
    ArrayDbl temp = Tree[connectingNode-1];
    double xer = (1/DIV)*(Route_Tree[Route_Count][0] - temp[0]);
    double yer = (1/DIV)*(Route_Tree[Route_Count][1] - temp[1]);
    for(int i = 1; i <= DIV-1; i++) { // divides up each node length
        Route_Count = Route_Count + 1;
        Route_Tree[Route_Count][0] = Route_Tree[Route_Count - 1][0] -
xer;
        Route_Tree[Route_Count][1] = Route_Tree[Route_Count - 1][1] -
yer;
    }
    Route_Count = Route_Count + 1;
}

```

```

        // MOVES THE PREVIOUS NODE IN THE TREE INTO THE FINAL
TREE
        Route_Tree[Route_Count][0] = temp[0];
        Route_Tree[Route_Count][1] = temp[1];
        // SETS THE NEXT CONNECTING NODE
        connectingNode = temp[2];
    }

    // cout<<"Route Tree"<<endl;
    // for(int i = 0; i < 10; i++) {
        // cout<<Route_Tree[i][0]<< " "<<Route_Tree[i][1]<<endl;
    // }
    // cout<<endl;

    // Initialize Path and Counters
    int nn = 0;
    int cc = Route_Count;
    // Start at the Rout-Div element of the route
    tests = (Route_Count+1) - divLook;
    // Initial position of the vehicle based on LiDAR Coords
    Path[0][0] = 0;
    Path[0][1] = 0;
    Path[0][2] = 0;

    // For each point in ROUTE_TREE-DIV_LOOK, estimate a curve from the
current
    // point to the DIV_LOOKth point and take a small step along that curve.
    for(int i = tests; i >= 1; --i) {
        // Make sure that the path is still valid
        if(nogogo == false) {
            // CALCULATE Lfw distance from current position to some node
Div_Look ahead
            LFW = sqrt( pow( Path[nn][0] - Route_Tree[cc - divLook][0] , 2 )
+ pow( Path[nn][1] - Route_Tree[cc-divLook][1] , 2 ) );
            // CALCULATE Nu The angle between current heading and the
node that
            // is Div_look ahead Recall that Theta and Nu are negative in
Quadrant 1 and
            // Positive in Quadrant 2.
            nu = -atan( ( Route_Tree[cc-divLook][0] - Path[nn][0] )/(
Route_Tree[cc-divLook][1] - Path[nn][1] ) ) - Path[nn][2];
            // Check feasibility of turn
            if( abs(nu) > M_PI/2 )
                nogogo = true;
            // Update counter for route tree
            cc = cc - 1;
        }
    }
}

```

```

// if the node is directly ahead of the car and no turn is needed
if(nu == 0) {
    radius = 0;
    // Update Path to send car directly to that point
    nn = nn + 1;
    Path[nn][0] = Route_Tree[cc][0];
    Path[nn][1] = Route_Tree[cc][1];
    // No change in theta if the car does not turn
    Path[nn][2] = Path[nn-1][2];
    Path[nn][3] = 0;
    // Estimate time needed to complete maneuver
    timeDiff = ( pow( pow( Path[nn][0] - Path[nn-1][0], 2) +
    pow( Path[nn][1] - Path[nn-1][1], 2), 0.5 ) )/Vehicle[3];
    timeTot = timeTot + timeDiff;
    Path[nn][4] = timeTot;
    //Path[nn][5] = radius;
}
else { // Turn required
    // Get Current orientation of vehicle
    theta = Path[nn][2];
    // Radius of turn should be signed
    radius = LFW/(2*sin(nu));
    // Angle of inscribed chord
    phi = 2*nu;
    // Location of the center of turn
    center[0] = Path[nn][0] - radius*cos(theta);
    center[1] = Path[nn][1] - radius*sin(theta);
    // Calculate steer angle needed to follow the given curve,
this
    // approximation uses ackerman steer for a bicycle model as
an
    // overly simplistic approximation of the steer angle, future
models
    // developments should add slip angle as well as dynamic
    Path[nn][3] = atan(length/radius)*(180/M_PI);
    // Check that steer angle is within physical limitations of
    // hardware
    if(abs(Path[nn][3]) > angleLimit)
        nogogo = true;

    // Next timestep
    nn = nn + 1;
    // add next node to tree along the arc [x,y,theta]
    // if statement accounts for the case where the car is turning
    // right and the next node is to the left and vice versa
    // Angle used to find x,y of next step

```

```

        double alpha = (phi/angleDiff) + theta;
        // find next x,y point
        Path[nn][0] = center[0] + radius*cos(alpha);
        Path[nn][1] = center[1] + radius*sin(alpha);
        // Find heading at next step for perfect geometric steering
        Path[nn][2] = (phi/angleDiff) + theta;
        // Check that the NN node is not behind the NN-1 Node
        if( Path[nn][1] <= Path[nn-1][1] )
            nogogo = true;

        // Estimate time needed to complete manuver
        timeDiff = abs( (phi/angleDiff)*(radius/Vehicle[3]) );
        timeTot = timeTot + timeDiff;
        Path[nn][4] = timeTot;
        pathCount = nn;
    }
}

// If any part of the path was not achieved, set the fail flag to
// try a new set of nodes
if( nogogo == true) {
    Path[0][0] = 9001;
    break;
}
}

bool RRT::CheckCollision(ArrayDbl Obstacle, float Vehicle_Geo[],ArrayDbl Road) {
    int nn = 0;
    float length = Vehicle_Geo[0];
    float width = Vehicle_Geo[1];
    double V_ObstacleX = Obstacle[4];           // VELOCITY OF OBSTACLE in X
    direction [ft/s]
    double V_ObstacleY = Obstacle[5];           // VELOCITY OF OBSTACLE in Y
    direction [ft/s]
    bool NOGOGO = false;
    double ObstacleB[2][2] ;
    memset( ObstacleB,0,sizeof(ObstacleB) );

    double m = Road_Lines[0];                   // slope
    double Bl = Road_Lines[1];                  // left road boundary y-intercept
    double Br = Road_Lines[2];                  // right road boundary y-intercept
    double yaw = Road[2];

    // OBSTACLE
    double Center_Pos[] = { Obstacle[0], Obstacle[1] }; // [x,y]
    double O_HIT = Obstacle[3];                  // HEIGHT[ft]
    double O_WIT = Obstacle[2];                  // WIDTH [ft]
}

```

```

// INITIAL POSITION
double OX1 = Center_Pos[0] - (O_WIT/2); // LEFT BOUNDARY
%THE OBSTACLE IS DEFINED BY A RECTANGLE
double OX2 = Center_Pos[0] + (O_WIT/2); // RIGHT BOUNDARY
    %THESE VALUES WILL BE COMING FROM SENSORS
double OY1 = Center_Pos[1] + (O_HIT/2); // UPPER BOUNDARY
%AND WILL BE SUBJECT TO CHANGE WHEN THE OBSTACLE MOVES
double OY2 = Center_Pos[1] - (O_HIT/2); // LOWER BOUNDARY

double Diag = sqrt( pow( Vehicle_Geo[0], 2 )+ pow( Vehicle_Geo[1], 2 ));

// THE CORNER TO CORNER LENGTH OF THE VEHICLE, CONSTANT

while ( (Path[nn+1][4] > 0) && (NOGOGO == false) && ((nn+1) < 50) ) {

    nn += 1;

    double x_0= Path[nn][0];                                // center x-
position of car
    double y_0= Path[nn][1];                                // center y-
position of car
    double theta = Path[nn][2]*(M_PI/180);                // angle of car relative
to Lidar-Coordinate system
    double Theta0 = atan((length/2)/(width/2)); // angle from center of car to
vertices

    // CALCULATE VERTICES OF THE CAR
    // VERTICES
    // 1=front left corner
    // 2=front right corner
    // 3=back left corner
    // 4=back right corner
    double Theta14 = Theta0 - theta; // Angle from vertices 1 and 4 from
lidar coordinate system
    double Theta23 = Theta0 + theta; // Angle from vertices 2 and 3 from
lidar coordinate system

    double dx14 = Diag/2*cos(Theta14); // Change in x-position from
parallel to coordinate system for 1 and 4
    double dx23 = Diag/2*cos(Theta23); // Change in x-position from
parallel to coordinate system for 2 and 3
    double dy14= Diag/2*sin(Theta14); // Change in y-position from parallel
to coordinate system for 1 and 4
    double dy23= Diag/2*sin(Theta23); // Change in y-position from parallel
to coordinate system for 2 and 3

```

```

double Vertices[4][2];
Vertices[0][0] = x_0 - dx14;           // front left corner
Vertices[0][1] = y_0 + dy14;
Vertices[1][0] = x_0 - dx23;           // back left corner
Vertices[1][1] = y_0 - dy23;
Vertices[2][0] = x_0 + dx23;           // front right corner
Vertices[2][1] = y_0 + dy23;
Vertices[3][0] = x_0 + dx14;           // back right corner
Vertices[3][1] = y_0 - dy14;

// CHECK TO SEE IF VERTICES ARE OFF THE ROAD
if(yaw == 0) {                         // STRAIGHT
ROAD SCENARIO
    for(int i=0; i < 4; i++) {
        if( (Vertices[i][0] < Road[0])||(Vertices[i][0] > Road[1])) {
            NOGOGO = true;
            break;
        }
    }
}
else {                                  // ANGLED
ROAD SCENARIOS
    double Y_max[4];
    double Y_min[4];
    // Finds the y values on the road based on the x-positions of the
vertices of the car
    if(yaw > 0) { // road veers left
        Y_max[0] = Vertices[0][0]*m + Bl; Y_max[1] =
Vertices[1][0]*m + Bl;
        Y_max[2] = Vertices[2][0]*m + Bl; Y_max[3] =
Vertices[3][0]*m + Bl;
        Y_min[0] = Vertices[0][0]*m + Br; Y_min[1] =
Vertices[1][0]*m + Br;
        Y_min[2] = Vertices[2][0]*m + Br; Y_min[3] =
Vertices[3][0]*m + Br;
    }
    if(yaw < 0) { //road veers right
        Y_max[0] = Vertices[0][0]*m + Br; Y_max[1] =
Vertices[1][0]*m + Br;
        Y_max[2] = Vertices[2][0]*m + Br; Y_max[3] =
Vertices[3][0]*m + Br;
        Y_min[0] = Vertices[0][0]*m + Bl; Y_min[1] =
Vertices[1][0]*m + Bl;
        Y_min[2] = Vertices[2][0]*m + Bl; Y_min[3] =
Vertices[3][0]*m + Bl;
    }
}

// CHECK Y VALUES

```

```

        // makes sure that none of the y-values of the vertices of the car are
        above or below the road
        for(int i=0; i < 4; i++) {
            if( (Y_max[i]-Vertices[i][1] < 0) || (Vertices[i][1]-Y_min[i])
            < 0) {
                NOGOGO = true;
                break;
            }
        }
        // CHECK IF PATH AVOIDS OBSTACLE
        double Time_Tot = Path[nn][4];

        // SET THE OBSTACLE BOUNDING BOX
        ObstacleB[0][0] = OX1 + V_ObstacleX*Time_Tot; // LEFT SIDE X
        ObstacleB[0][1] = OY1 + V_ObstacleY*Time_Tot; // FURTHER SIDE
        Y
        ObstacleB[1][0] = OX2 + V_ObstacleX*Time_Tot; // RIGHT SIDE X
        ObstacleB[1][1] = OY2 + V_ObstacleY*Time_Tot; // CLOSER SIDE Y

        // CHECK COLLISION WITH OBSTACLE
        double maxX =
        fmax(fmax(fmax(Vertices[0][0], Vertices[1][0]), Vertices[2][0]), Vertices[3][0]);
        double minX =
        fmin(fmin(fmin(Vertices[0][0], Vertices[1][0]), Vertices[2][0]), Vertices[3][0]);
        double maxY =
        fmax(fmax(fmax(Vertices[0][1], Vertices[1][1]), Vertices[2][1]), Vertices[3][1]);
        double minY =
        fmin(fmin(fmin(Vertices[0][1], Vertices[1][1]), Vertices[2][1]), Vertices[3][1]);
        if(NOGOGO == false) {
            if(minX > ObstacleB[1][0]) { // THE VEHICLE IS
ON THE RIGHT SIDE OF THE OBSTACLE

            }
            else if(maxX < ObstacleB[0][0]) { // THE VEHICLE IS ON
THE LEFT SIDE OF THE OBSTACLE

            }
            else if(maxY < ObstacleB[1][1]) { // THE VEHICLE IS
BEFORE THE OBSTACLE

            }
            else if(minY > ObstacleB[0][1]) { // THE VEHICLE IS PAST
THE OBSTACLE

            }
        }
    }
}

```

```

        else {
            NOGOGO = true;
        }
    }

    return NOGOGO;
}

root/src/BeagleBone/source/Segmentation/segmentation_class.cpp

/*
 * segmentation_class.cpp
 *
 * Created on: Apr 16, 2014
 *      Author: Thomas
 */
//segmentation_class.cpp
#include "segmentation_class.h"
template <typename T> int sgn(T val) {
    return (T(0) < val) - (val < T(0));
}
Segmentation::Segmentation() {
    Threshold = 12; maxNumPts = 75; MaxVelocity = 15.0; ScalingFactor = 1.1;
    BEAMNUMBER = 682;
    angularRes_rad = 0.36*(M_PI/180.0); areamin = 0.00001; noRoadCount = 0;
    noRoadFlag = false; prevRoadSlope = 0; deltaYaw = 0; currentYaw =
    0.0000001; prevYaw = 0.0000001;
    rotateIndices = 0; maxSlopeChange = 30; areaThreshold = 0.15; roadWidth =
    10.0;//12.3;
    areaBuffer = 0.125; coneSpacing = 4.0; spacingBuffer = 0.035;
    coneRoadBoundOffset = 1.0; maxStepSize = 1.0;
    threatThreshold = 1000065; distThreshold = 10; prevToggle = 0; Toggle = 0;
    minSpeed = -1.0;
    minDistance = 0.30; threatConeAngle = 45; allowableDistToCone = 2.8;
    thresholdManueverAngle = 10;
    emergencyDistToCone = 4.25;
    for(int i = 0;i < 6; i++) {
        roadBounds[i] = 0;
        prevRoadBounds[i] = 0;
    }
    feasibleRoadway = false;
}
Segmentation::Segmentation(int Threshold, int maxNumPts, double ScalingFactor,
double roadWidth, double areaThreshold,

```

```

        double coneSpacing, double areaBuffer, double spacingBuffer, double
coneRoadBoundOffset, double threatThreshold, double distThreshold,
        double minSpeed, double minDistance, double threatConeAngle, double
allowableDistToCone, double thresholdManueverAngle, double maxStepSize):
    Threshold(Threshold), maxNumPts(maxNumPts),
ScalingFactor(ScalingFactor), roadWidth(roadWidth), areaThreshold(areaThreshold),
        coneSpacing(coneSpacing), areaBuffer(areaBuffer),
spacingBuffer(spacingBuffer), coneRoadBoundOffset(coneRoadBoundOffset),
threatThreshold(threatThreshold),
        distThreshold(distThreshold), minSpeed(minSpeed),
minDistance(minDistance), threatConeAngle(threatConeAngle),
allowableDistToCone(allowableDistToCone),
        thresholdManueverAngle(thresholdManueverAngle),
maxStepSize(maxStepSize)
    { BEAMNUMBER = 682; feasibleRoadway = false; }

Segmentation::Segmentation(int Threshold, int maxNumPts,double ScalingFactor,double
roadWidth,double areaThreshold,
        double coneSpacing,double areaBuffer,double spacingBuffer,double
coneRoadBoundOffset,double threatThreshold,double distThreshold,
        double minSpeed,double minDistance,double threatConeAngle,double
allowableDistToCone,double thresholdManueverAngle,double maxStepSize, int
BEAMNUMBER):

    Threshold(Threshold),maxNumPts(maxNumPts),ScalingFactor(ScalingFactor),ro
adWidth(roadWidth),areaThreshold(areaThreshold),
        coneSpacing(coneSpacing),areaBuffer(areaBuffer),spacingBuffer(spacingBuffer),
coneRoadBoundOffset(coneRoadBoundOffset),threatThreshold(threatThreshold),
        distThreshold(distThreshold),minSpeed(minSpeed),minDistance(minDistance),thr
eatConeAngle(threatConeAngle),allowableDistToCone(allowableDistToCone),
        thresholdManueverAngle(thresholdManueverAngle),maxStepSize(maxStepSize),
BEAMNUMBER(BEAMNUMBER)
    { feasibleRoadway = false; }

// convert double[] to vectors and get rid of beamnumber
void Segmentation::identifyObstacles(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration) {
    // Cluster/Group Points within Point-Cloud
    clusterData(x,y,prev_x,prev_y,scanDuration);
    // Segment clustered data
    segmentData();
    // Parse/differentiate/Identify round bounds
    parseRoadBounds();
    // Update obstacles with true velocity after coordinate rotation determination and
reprocess obstacles if car has turned
}

```

```

updateRoadBounds();

// if(deltaYaw!=0) {
    // Sort data and clear buffers
    // AllObstacles.clear();
    // Obstacle.clear();
    // PossibleRoadPts.clear();
    // toRemove.clear();
    // RoadPts.clear();
    // updateCorrectAdjustedVelocities(x,y,prev_x,prev_y,scanDuration);
// }
if(noRoadFlag==true || feasibleRoadway==false)
    Toggle = -1;
else if(Toggle!=-1)
    determineThreat( new double[2] { 0, 15 } );

//
// cout<< "All Obstacles"<<endl;
// for(unsigned int i=0;i<AllObstacles.size();i++) {
//     ArrayDbl temp = AllObstacles[i];
//     for(unsigned int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
//     cout<<endl;
// }
// cout<<endl;

//
// cout<<"Threatening Obstacles"<<endl;
// for(unsigned int i=0;i<ThreateningObstacles.size();i++) {
//     ArrayDbl temp = ThreateningObstacles[i];
//     for(unsigned int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
//     cout<<endl;
// }
// cout<<endl;

//
ArrayDbl getRoadBounds = returnRoadBounds();
for(unsigned int i=0; i < getRoadBounds.size(); i++) {
    cout<<getRoadBounds[i]<<" ";
}
cout<<endl;
ReturnAllObstacles = AllObstacles;
ReturnObstacles = ThreateningObstacles;
ReturnThreateningMarkers = ThreateningMarkers;

AllObstacles.clear();
Obstacle.clear();
PossibleRoadPts.clear();
toRemove.clear();
RoadPts.clear();

```

```

ThreateningObstacles.clear();
ThreateningMarkers.clear();
}
void Segmentation::identifyObstacles(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration, double vehicleState[], int beamNumber) {
    BEAMNUMBER = beamNumber;
    // Cluster/Group Points within Point-Cloud
    clusterData(x,y,prev_x,prev_y,scanDuration);
    // Segment clustered data
    segmentData();
    // Parse/differentiate/Identify round bounds
    parseRoadBounds();
    // Update obstacles with true velocity after coordinate rotation determination and
    reprocess obstacles if car has turned
    updateRoadBounds();

    // if(deltaYaw!=0) {
        // Sort data and clear buffers
        // AllObstacles.clear();
        // Obstacle.clear();
        // PossibleRoadPts.clear();
        // toRemove.clear();
        // RoadPts.clear();
        // updateCorrectAdjustedVelocities(x,y,prev_x,prev_y,scanDuration);
    // }
    if(noRoadFlag==true || feasibleRoadway==false)
        Toggle = -1;
    else {
        determineThreat(vehicleState);
        // no wheel encoders
        // determineThreat( new double[2] { 0, 2 } );
    }

    // cout<< "All Obstacles"<<endl;
    // for(unsigned int i=0;i<AllObstacles.size();i++) {
    //     ArrayDbl temp = AllObstacles[i];
    //     for(unsigned int j=0;j<temp.size();j++) {      cout<<temp[j]<<" "; }
    //     cout<<endl;
    // }
    // cout<<endl;
    // cout<<"Threatening Obstacles"<<endl;
    // for(int i=0;i<ThreateningObstacles.size();i++) {
        // ArrayDbl temp = ThreateningObstacles[i];
        // for(int j=0;j<temp.size();j++) {      cout<<temp[j]<<" "; }
        // cout<<endl;
    // }

```

```

//      ArrayDbl getRoadBounds = returnRoadBounds();
//      for(unsigned int i=0; i < getRoadBounds.size(); i++) {
//          cout<<getRoadBounds[i]<<" ";
//      }

ReturnAllObstacles = AllObstacles;
ReturnObstacles = ThreateningObstacles;
ReturnThreateningMarkers = ThreateningMarkers;

AllObstacles.clear();
Obstacle.clear();
PossibleRoadPts.clear();
toRemove.clear();
RoadPts.clear();
ThreateningObstacles.clear();
ThreateningMarkers.clear();
}
vector< vector<double> > Segmentation::getObstacleArray() {
    return ReturnObstacles;
}
vector< vector<double> > Segmentation::getAllObstacleArray() {
    return ReturnAllObstacles;
}
void Segmentation::determineThreat(double vehicleState[]) {
    int count = 0;
    threat_type = NONE;
    threat_region = NA;
    double VehicleSpeed = sqrt(pow(vehicleState[0],2)+pow(vehicleState[1],2));
    if((VehicleSpeed > minSpeed)&&(roadBounds[0]!=0)&&(roadBounds[1]!=0)) {
        for(unsigned int i = 0; i<AllObstacles.size();i++) {
            ArrayDbl temp = AllObstacles[i];
            double tempDist = sqrt(pow(temp[0],2)+pow(temp[1],2));
            if(tempDist>=minDistance) {
                double deltaTx = abs(temp[4]/tempDist);
                double deltaTy = abs(temp[5]/tempDist);
                double deltaT = min(deltaTx,deltaTy);
                double threat = 0;
                if(deltaT != 0)
                    threat = 1/deltaT;
                if((threat>=threatThreshold)||((tempDist<=distThreshold)) {
                    ArrayDbl threatAdd;
                    threatAdd.push_back(temp[0]); threatAdd.push_back(temp[1]);
                    threatAdd.push_back(temp[2]);
                    threatAdd.push_back(temp[3]);
                    // threatAdd.push_back(temp[4]+vehicleState[0]);
                    // threatAdd.push_back(temp[5]+vehicleState[1]);
                }
            }
        }
    }
}

```

```

        threatAdd.push_back(temp[4]);
        threatAdd.push_back(temp[5]);
        // Check threat cone
        int insideLeftLine = sgn( temp[1] - tan(M_PI-
threatConeAngle*(M_PI/180))*temp[0] );
        int insideRightLine = sgn( temp[1] -
tan(threatConeAngle*(M_PI/180))*temp[0] );
        if( insideLeftLine>=0 && insideRightLine>=0 &&
temp[1] > 0) { // Check if obstacle is inside threat cone
        Toggle = 1;

        ThreateningObstacles.push_back(threatAdd);
        threat_type = OBSTACLE;
    }
}
}

if(Toggle<1) {
    double distToLeftBound =
abs(roadBounds[4])/sqrt(pow(roadBounds[3],2)+1);
    double distToRightBound =
abs(roadBounds[5])/sqrt(pow(roadBounds[3],2)+1);

    double smallestDistToBound;
    if(distToLeftBound < distToRightBound)
        smallestDistToBound = distToLeftBound;
    else
        smallestDistToBound = distToRightBound;

    if(smallestDistToBound <= emergencyDistToCone) {
        // #ifdef ENABLE_LKA
        Toggle = 1;
        // #endif
        threat_type = ROADBOUNDARY;
        threat_region = HIGH;
    } else
if((abs(currentYaw*180/M_PI)>=thresholdManueverAngle)&&(smallestDistToBound<=
allowableDistToCone)&&roadBoundaryYawThreat_possible) {
        // #ifdef ENABLE_LKA
        Toggle = 1;
        // #endif
        threat_type = ROADBOUNDARY;
        threat_region = MEDIUM;
    } else
if((prevToggle==1)&&(abs(currentYaw*180/M_PI)>=thresholdManueverAngle)&&roa
dBoundaryYawThreat_possible) {

```

```

        // #ifdef ENABLE_LKA
        Toggle = 1;
    // #endif
    threat_type = ROADBOUNDARY;
    threat_region = MEDIUM;
}
}

// Don't stop autonomous control in middle of autonomous maneuver
//
if((Toggle==0)&&(prevToggle==1)&&(abs(currentYaw*180/M_PI)>=thresholdManeuverAngle)&&roadBoundaryYawThreat_possible)
    if((Toggle<1)&&(prevToggle>0)) {
        prevToggle = 0;
        Toggle = 1;
        threat_type = RRT_MANUEVER;
    }
    else {
        prevToggle = Toggle;
    }
}
void Segmentation::clusterData(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration) {
    double previousVal = 0;
    for(int i=0; i<BEAMNUMBER; i++) {
        double mag = sqrt(pow(x[i],2)+pow(y[i],2));
        double step = 0;
        if(i==0) { previousVal = mag; }
        else { step = sqrt(pow(x[i-1]-x[i],2)+pow(y[i-1]-y[i],2)); }
        if(mag<=Threshold) {
            ArrayDbl temp;
            temp.push_back(x[i]); temp.push_back(y[i]);
            temp.push_back(prev_x[i]); temp.push_back(prev_y[i]);
            Obstacle.push_back(temp);

            if((Obstacle.size()>maxNumPts)||((step>maxStepSize)))//&&(Obstacle.size()>1))
                // TODO: add parameter minNumPts (default = 2)? what about when size == 1
                getObstacleInfo(mag,scanDuration);
        }
        else {
            // If outside threshold and last point was inside threshold (and part
            // of obstacle def.) end current obstacle def. or cluster
            if(previousVal<=Threshold) { getObstacleInfo(mag,scanDuration); }
        }
    }
}

```

```

        previousVal = mag;
    }
}

void Segmentation::getObstacleInfo(double distance,double deltaT) {
    int j;
    ArrayDbl temp;
    double xmin=0,xmax=0,ymin=0,ymax=0,VxPt=0,VyPt=0;
    int lastIt = Obstacle.size();
    if(Obstacle.size()>0) {
        for(j=0;j<lastIt;j++) {
            temp = Obstacle[j];
            if(j==0) { xmin=temp[0]; xmax=temp[0]; ymin=temp[1];
            ymax=temp[1]; VxPt=temp[2]; VyPt=temp[3]; }
            if(temp[0]<xmin)
                xmin = temp[0];
            if(temp[0]>xmax)
                xmax = temp[0];
            if(temp[1]<ymin)
                ymin = temp[1];
            if(temp[1]>ymax)
                ymax = temp[1];
            if((j!=0)&&(j!=lastIt)) {
                VxPt = VxPt+chkMaxVelocity((temp[2]-temp[0])/deltaT,
                VxPt,j);
                VyPt = VyPt+chkMaxVelocity((temp[3]-temp[1])/deltaT,
                VyPt,j);
            }
            temp.clear();
        }
        double xObstacle = (xmin+xmax)/2;
        double yObstacle = (ymin+ymax)/2;
        double dx = getBoxLength(xmin, xmax, distance*angularRes_rad);
        double dy = getBoxLength(ymin, ymax, distance*angularRes_rad);
                    // Get obstacle length
        double Vx = 0.0;//VxPt/Obstacle.size();
                    // Get obstacle x velocity
        double Vy = 0.0;//VyPt/Obstacle.size();
                    // Get obstacle y velocity
        double tempData[] = {xObstacle, yObstacle, dx, dy, Vx, Vy};
        vector<double> ObstacleData(tempData,
        tempData+sizeof(tempData)/sizeof(double));
        AllObstacles.push_back(ObstacleData);
                    // Add obstacle data array to
array containing all obstacles
        Obstacle.clear();
    }
}

```

```

}

void Segmentation::segmentData() {
    for(unsigned int i=0;i<AllObstacles.size();i++) {
        ArrayDbl temp = AllObstacles[i];
        double tempArea = temp[2]*temp[3];
        if((tempArea<=areaThreshold)&&(tempArea>=areamin))
            PossibleRoadPts.push_back( i );
    }
}

void Segmentation::parseRoadBounds() {
    for(unsigned int i=0;i<PossibleRoadPts.size();i++) {
        int index1 = PossibleRoadPts[i];
        ArrayDbl compare1 = AllObstacles[index1];
        for(unsigned int j=0;j<PossibleRoadPts.size();j++) {
            if(j==i)
                continue;
            else {
                int index2 = PossibleRoadPts[j];
                ArrayDbl compare2 = AllObstacles[index2];
                double spacingDist = sqrt( pow(compare2[0]-
compare1[0],2)+pow(compare2[1]-compare1[1],2) );
                if((spacingDist>=(coneSpacing-
coneSpacing*spacingBuffer))&&(spacingDist<=(coneSpacing+coneSpacing*spacingBuf
fer))) {
                    ArrayDbl temp;
                    temp.push_back(compare1[0]);
                    temp.push_back(compare1[1]);
                    RoadPts.push_back(temp);
                    temp.clear();
                    temp.push_back(compare2[0]);
                    temp.push_back(compare2[1]);
                    RoadPts.push_back(temp);
                    toRemove.push_back(index1);
                    toRemove.push_back(index2);
                }
            }
        }
    }
    sort( toRemove.begin(), toRemove.end() );
    toRemove.erase( unique( toRemove.begin(), toRemove.end() ), toRemove.end() );
    for(unsigned int i=0; i < RoadPts.size(); i++) {
        ArrayDbl compare1 = RoadPts[i];
        for(unsigned int j=0; j < RoadPts.size();j++) {
            if(j==i)
                continue;
            else {

```

```

        ArrayDbl compare2 = RoadPts[j];

        if((compare1[0]==compare2[0])&&(compare1[1]==compare2[1])) {
            RoadPts.erase(RoadPts.begin() + j);
            j=j-1;
        }
    }
}

removeRoadCones();
}

void Segmentation::removeRoadCones() {
    int remove = 0;
    for(unsigned int i=0; i < toRemove.size(); i++) {
        try {
            int index = toRemove[i];
            AllObstacles.erase(AllObstacles.begin() + (index - remove));
            remove++;
        } catch(...) { }
    }
}

void Segmentation::updateRoadBounds() {
    for(int i=0; i < 6; i++) {
        roadBounds[i] = 0;
    }
    double rightBound1[5] = {-1,0,0,0,0};
    double rightBound2[5] = {-1,0,0,0,0};
    double leftBound1[5] = {-1,0,0,0,0};
    double leftBound2[5] = {-1,0,0,0,0};
    int roadFlag = 0;

    for(unsigned int i=0; i < RoadPts.size(); i++) {
        ArrayDbl temp = RoadPts[i];
        double tempDist = sqrt( pow( temp[0],2 ) + pow( temp[1],2 ) );
        double theta = atan2( temp[1], temp[0] )*(180/M_PI);
        if(temp[0] >= 0) {
            if(rightBound1[0] >= 0) {
                double spacingDist = sqrt( pow( rightBound1[3] - temp[0],2 ) + pow( rightBound1[4] - temp[1],2 ) );
                if((spacingDist >
                    (coneSpacing+coneSpacing*spacingBuffer))&&(spacingDist<(coneSpacing-
                    coneSpacing*spacingBuffer))) {
                    if(theta < rightBound1[1]) {
                        rightBound1[0] = i;
                        rightBound1[1] = theta;
                        rightBound1[2] = tempDist;
                    }
                }
            }
        }
    }
}

```

```

        rightBound1[3] = temp[0];
        rightBound1[4] = temp[1];
    }
}
else {
    rightBound2[0] = i;
    rightBound2[1] = theta;
    rightBound2[2] = tempDist;
    rightBound2[3] = temp[0];
    rightBound2[4] = temp[1];
}
}
else {
    rightBound1[0] = i;
    rightBound1[1] = theta;
    rightBound1[2] = tempDist;
    rightBound1[3] = temp[0];
    rightBound1[4] = temp[1];
}
}
else {
    if(leftBound1[0] >= 0) {
        double spacingDist = sqrt( pow(leftBound1[3] - temp[0],2 ) +
+ pow(leftBound1[4] - temp[1],2 ) );
        if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
            if(theta > leftBound1[1]) {
                leftBound1[0] = i;
                leftBound1[1] = theta;
                leftBound1[2] = tempDist;
                leftBound1[3] = temp[0];
                leftBound1[4] = temp[1];
            }
        }
        else {
            leftBound2[0] = i;
            leftBound2[1] = theta;
            leftBound2[2] = tempDist;
            leftBound2[3] = temp[0];
            leftBound2[4] = temp[1];
        }
    }
    else {
        leftBound1[0] = i;
        leftBound1[1] = theta;
    }
}
}

```

```

        leftBound1[2] = tempDist;
        leftBound1[3] = temp[0];
        leftBound1[4] = temp[1];
    }
}
if((rightBound1[0] >= 0)&&(rightBound2[0] >= 0)) {
    double spacingDist = sqrt( pow( rightBound1[3] -
rightBound2[3],2 ) + pow( rightBound1[4] - rightBound2[4],2 ) );
    if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
        if(rightBound2[1] > rightBound1[1]) {
            rightBound1[0] = rightBound2[0];
            rightBound1[1] = rightBound2[1];
            rightBound1[2] = rightBound2[2];
            rightBound1[3] = rightBound2[3];
            rightBound1[4] = rightBound2[4];
        }
    }
    else {
        roadFlag = 1;
        break;
    }
}
else if((leftBound1[0] >= 0)&&(leftBound2[0] >= 0)) {
    double spacingDist = sqrt( pow(leftBound1[3] - leftBound2[3],2 )
+ pow(leftBound1[4] - leftBound2[4],2 ) );
    if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
        if(leftBound2[1] < leftBound1[1]) {
            leftBound1[0] = leftBound2[0];
            leftBound1[1] = leftBound2[1];
            leftBound1[2] = leftBound2[2];
            leftBound1[3] = leftBound2[3];
            leftBound1[4] = leftBound2[4];
        }
    }
    else {
        roadFlag = 2;
        break;
    }
}
}

double m = 0,d_l,d_r,yaw,offset = 0,b_r = 0,b_l = 0;

```

```

bool slopeCheck = true;
double *roadCalc;
roadBoundaryYawThreat_possible = false;
if(roadFlag == 1) {
    if(rightBound2[2] > rightBound1[2])
        roadCalc = calculateSlopeIntercept(rightBound1,rightBound2);
    else
        roadCalc = calculateSlopeIntercept(rightBound2,rightBound1);
    m = (*roadCalc); b_r = (*(roadCalc+1));
    // if(prevRoadSlope != 0)
        // slopeCheck = checkRoadSlope();
    yaw = atan(1/m);
    if(yaw == 0) { yaw = 0.0000001; }
    slopeCheck = checkRoadSlope(yaw);

    if(slopeCheck == false) {

        double tempSlope = m;
        m = prevRoadSlope;
        prevRoadSlope = tempSlope;

        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {
            roadBounds[i] = 0;
        }
    } else {
        prevYaw = currentYaw;
        currentYaw = yaw;

        prevRoadSlope = m;
        offset = roadWidth/sin(yaw);
        d_r = -b_r/m;
        b_l = b_r + offset;
        d_l = -b_l/m;
        roadBounds[0] = d_l+coneRoadBoundOffset;
        roadBounds[1] = d_r-coneRoadBoundOffset;
        roadBounds[2] = yaw*(180/M_PI);
        roadBounds[3] = m;
        roadBounds[4] = b_l;
        roadBounds[5] = b_r;
        memcpy(prevRoadBounds,roadBounds,6 * sizeof(*roadBounds));

        if(abs(d_l) < abs(d_r)) {
            if(b_l > 0) {
                roadBoundaryYawThreat_possible = true;
            }
        }
    }
}

```

```

        } else {
            if(b_r > 0) {
                roadBoundaryYawThreat_possible = true;
            }
        }
    }

cleanUpObstacles();
delete[] roadCalc;

noRoadCount = 0;
noRoadFlag = false;
Toggle = 0;
}
else if(roadFlag == 2) {
    if(leftBound2[2] > leftBound1[2])
        roadCalc = calculateSlopeIntercept(leftBound1,leftBound2);
    else
        roadCalc = calculateSlopeIntercept(leftBound2,leftBound1);
    m = (*roadCalc); b_l = (*roadCalc+1));
    yaw = atan(1/m);
    if(yaw == 0) { yaw = 0.0000001; }
    slopeCheck = checkRoadSlope(yaw);

    if(slopeCheck == false) {

        double tempSlope = m;
        m = prevRoadSlope;
        prevRoadSlope = tempSlope;

        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {
            roadBounds[i] = 0;
        }
    } else {
        prevYaw = currentYaw;
        currentYaw = yaw;

        prevRoadSlope = m;
        offset = roadWidth/sin(yaw);
        d_l = -b_l/m;
        b_r = b_l - offset;
        d_r = -b_r/m;
        roadBounds[0] = d_l+coneRoadBoundOffset;
        roadBounds[1] = d_r-coneRoadBoundOffset;
        roadBounds[2] = yaw*(180/M_PI);
    }
}

```

```

        roadBounds[3] = m;
        roadBounds[4] = b_l;
        roadBounds[5] = b_r;
        memcpy(prevRoadBounds,roadBounds,6 * sizeof(*roadBounds));

        if(abs(d_l) < abs(d_r)) {
            if(b_l > 0) {
                roadBoundaryYawThreat_possible = true;
            }
        } else {
            if(b_r > 0) {
                roadBoundaryYawThreat_possible = true;
            }
        }
    }

    cleanUpObstacles();
    delete[] roadCalc;

    noRoadCount = 0;
    noRoadFlag = false;
    Toggle = 0;
}
else {
    if((rightBound1[0] <
0)&&(rightBound2[0]<0)&&(leftBound1[0]<0)&&(leftBound2[0]<0))
        noRoadCount++;
    if(noRoadCount>3) {
        noRoadFlag = true;
        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {
            roadBounds[i] = 0;
        }
    } else {
        memcpy(roadBounds,prevRoadBounds,6 *
sizeof(*prevRoadBounds));
        currentYaw = atan(1/prevRoadSlope);
        feasibleRoadway = true;
        roadBoundaryYawThreat_possible = false;
    }
}
}

void Segmentation::cleanUpObstacles() {
    double br = -roadBounds[1]/tan(roadBounds[2]*(M_PI/180));
    double m = -br/roadBounds[1];
    double bl = -roadBounds[0]/tan(roadBounds[2]*(M_PI/180));
}

```

```

double rx1 = -100;
double ry1 = m*-100+br;
double rx2 = 100;
double ry2 = m*100+br;

double lx1 = -100;
double ly1 = m*-100+bl;
double lx2 = 100;
double ly2 = m*100+bl;

for(unsigned int i=0;i < AllObstacles.size(); i++) {
    ArrayDbl temp = AllObstacles[i];
    int leftofRightRoadline = sgn( -(rx2-rx1)*(temp[1]-ry1) - (ry2-
ry1)*(temp[0]-rx1) );
    int rightofLeftRoadline = sgn( (lx2-lx1)*(temp[1]-ly1) - (ly2-
ly1)*(temp[0]-lx1) );
    if(leftofRightRoadline != rightofLeftRoadline) {
        AllObstacles.erase(AllObstacles.begin() + i);
        i = i-1;
    }
}

// Check for feasible roadway (car within road boundaries)
int leftofRightRoadline = sgn( -(rx2-rx1)*(-ry1) - (ry2-ry1)*(-rx1) );
int rightofLeftRoadline = sgn( (lx2-lx1)*(-ly1) - (ly2-ly1)*(-lx1) );

if(leftofRightRoadline != rightofLeftRoadline) {
    feasibleRoadway = false;
    for(int i=0; i < 6; i++) {
        roadBounds[i] = 0;
    }
} else {
    feasibleRoadway = true;
}
}

bool Segmentation::checkRoadSlope(double yaw) {
    // deltaYaw = currentYaw - prevYaw;
    deltaYaw = yaw - prevYaw;
    if(abs(deltaYaw)*(180/M_PI) >= maxSlopeChange) {
        rotateIndices = 0;
        return false;
    }
} else {
    rotateIndices = -(deltaYaw/angularRes_rad);
    return true;
}

```

```
        }
    }
threat_type_t Segmentation::getThreatType() {
    return threat_type;
}
threat_region_t Segmentation::getThreatRegion() {
    return threat_region;
}
double* Segmentation::calculateSlopeIntercept(double closeBound[],double farBound[])
{
    double slope = 0;
    double intercept = 0;
    slope = ((farBound[4] - closeBound[4])/(farBound[3] - closeBound[3]));
    intercept = (closeBound[4] - slope*closeBound[3]);
    double *retArray = new double[2];
    *retArray = slope;
    *(retArray+1) = intercept;

    return retArray;
}
```

## Appendix C Hardware-in-the-Loop Simulation (VDHILS) Code

vdhils/Main.java:

```
/***
 * \file Main.java
 *      \brief Main is the starting point for the hardware in the loop simulator
 *      This class sets up and handles most GUI inputs from user. In addition,
 *      this class starts and sets the simulation state when user hits
 *      corresponding inputs.
 *
 *      \author Thomas Stevens
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  
package vdhils;
// Import Libraries
import vdhils.Graphics.*;
import vdhils.GUI.*;
import vdhils.Vehicle.CarParameters;
// GUI Imports
import javax.swing.*;
import javax.swing.plaf.LayerUI;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
```

```

import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Random;
import java.util.concurrent.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.io.IOException;
import java.beans.PropertyChangeEvent;
/**
 * \class Main
 * \brief Main class is entry point for application and handles the main actions
generated
 *
 * by user GUI and initializing simulation, update, and render classes.
*/
public class Main implements ActionListener {
    /**
     * Instance of Main class called from main (program entrance) */
    private static Main VDHILS;
    /**
     * Dimension of the screen display size in pixels */
    private final static Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();

    private static SimData simData;
    /**
     * Class that handles simulation update thread */
    private static SimUpdater simUpdater;
    /**
     * Class that handles simulation render thread */
    private static SimRenderer simRenderer;
    private BlockingQueue<BufferStrategy> bufferStrategyQueue;
    private final int width;
    private final int height;
    private JFrame mainFrame;
    private MainMenu mainMenuPanel;
    private SettingsMenu settingsMenu;
    private ParametersMenu parametersMenu;
    private HardwareInLoopSimulation sim;

    private JLayer<JPanel> jlayer;
}

```

```

private WaitLayerUI layerUI;
private final Timer stopper;

    private String versNumber;
public static void main(String[] args) {
    // Constructor kicks off the GUI
    VDHILS = new Main((int)screenSize.getWidth(), (int)screenSize.getHeight());
    // Start/run the simulation
    VDHILS.start();
}

class WaitLayerUI extends LayerUI<JPanel> implements ActionListener {
    private boolean mIsRunning;
    private boolean mIsFadingOut;
    private Timer mTimer;

    private int mAngle;
    private int mFadeCount;
    private int mFadeLimit = 15;

    @Override
    public void paint (Graphics g, JComponent c) {
        int w = c.getWidth();
        int h = c.getHeight();

        // Paint the view.
        super.paint (g, c);

        if (!mIsRunning) {
            return;
        }

        Graphics2D g2 = (Graphics2D)g.create();

        float fade = (float)mFadeCount / (float)mFadeLimit;
        // Gray it out.
        Composite urComposite = g2.getComposite();
        g2.setComposite(AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, .5f * fade));
        g2.fillRect(0, 0, w, h);
        g2.setComposite(urComposite);

        // Paint the wait indicator.
        int s = Math.min(w, h) / 5;
        int cx = w / 2;
        int cy = h / 2;

```

```

        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setStroke(new BasicStroke(s / 4, BasicStroke.CAP_ROUND,
BasicStroke.JOIN_ROUND));
        g2.setPaint(Color.white);
        g2.rotate(Math.PI * mAngle / 180, cx, cy);
        for (int i = 0; i < 12; i++) {
            float scale = (11.0f - (float)i) / 11.0f;
            g2.drawLine(cx + s, cy, cx + s * 2, cy);
            g2.rotate(-Math.PI / 6, cx, cy);

        g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
scale * fade));
    }

    g2.dispose();
}
public void actionPerformed(ActionEvent e) {
    if (mIsRunning) {
        firePropertyChange("tick", 0, 1);
        mAngle += 3;
        if (mAngle >= 360) {
            mAngle = 0;
        }
        if (mIsFadingOut) {
            if (--mFadeCount == 0) {
                mIsRunning = false;
                mTimer.stop();
            }
        }
        else if (mFadeCount < mFadeLimit) {
            mFadeCount++;
        }
    }
}

public void start() {
    if (mIsRunning) {
        return;
    }
    // Run a thread for animation.
    mIsRunning = true;
    mIsFadingOut = false;
    mFadeCount = 0;
    int fps = 24;
    int tick = 1000 / fps;
}

```

```

        mTimer = new Timer(tick, this);
        mTimer.start();
    }

    public void stop() {
        mIsFadingOut = true;
    }

    @Override
    public void applyPropertyChange(PropertyChangeEvent pce, JLayer l) {
        if ("tick".equals(pce.getPropertyName())) {
            l.repaint();
        }
    }
}

/**
 * Constructor for Main
 *
 * \param width The width of the program's inside portion of the frame
 * \param height The height of the program's inside portion of the frame
 */
public Main(final int width, final int height) {
    this.simData = new SimData(width, height);
    this.bufferStrategyQueue = new ArrayBlockingQueue<>(1);
    this.simRenderer = new SimRenderer(simData, bufferStrategyQueue,
width, height);
    this.simUpdater = new SimUpdater(simData);
    this.width = width;
    this.height = height;

    try {
        File versfile = new File("Info/VersionNumber.txt");
        FileReader fileReader = new FileReader(versfile);
        BufferedReader bufferedReader = new
BufferedReader(fileReader);
        String line = bufferedReader.readLine();
        if(line != null)
            versNumber = line;
        else
            versNumber = "";
    } catch(IOException e) {
        e.printStackTrace();
        simData.logger.severe("Unable to parse simulation version number
from CompilerInfo file");
    }
}

```

```

mainFrame = new JFrameWithResizeListener(simRenderer);
// This runnable will construct the GUI on the EDT, but also update our
// SimUpdater with a SwingComponentDrawer object
// that is created in this codebase, as well as store the created
// BufferStrategy to use for active rendering to the BlockingQueue
// passed.
    mainFrame.setAlwaysOnTop(true);
    mainMenuPanel = new MainMenu(this, width, height, versNumber);
    settingsMenu = new SettingsMenu(this, width, height, this.simData);
    parametersMenu = new ParametersMenu(this, width, height, this.simData);

Runnable guiCreator = new GuiCreatorRunnable(mainFrame, width, height,
simRenderer, bufferStrategyQueue, mainMenuPanel, versNumber);
try {
    // Invoke swing code on EDT thread (Swing code always executes
on EDT - this method ensures it)
    SwingUtilities.invokeAndWait(guiCreator);
} catch (InterruptedException | InvocationTargetException e) {
    e.printStackTrace();
    simData.logger.severe("Failed to create GUI on EDT (" +
e.getStackTrace() + ")");
}
layerUI = new WaitLayerUI();
stopper = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        layerUI.stop();
        simData.inputEnabled = true;
    }
});
stopper.setRepeats(false);
}

/**
 * Starts the rendering of the simulation including animation and movement
 */
void start() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            simRenderer.renderLoop();
        }
    }).start();
}

public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();

```

```

System.out.println("MainListener action: " + action);
simData.logger.info("MainListener action: " + action);
if(simData.inputEnabled) {
    simData.inputEnabled = false;
    if(action.equals("Start")) {
        // Start button pressed, initialize and launch new simulation
        CarParameters tempCarParams =
parametersMenu.getCarParameters();
        ArrayList<CarParameters> tempObstacleParams =
parametersMenu.getObstacleParameters();

        tempCarParams.setSerialPort(settingsMenu.getSerialPort());
        sim = new
HardwareInLoopSimulation(simUpdater,simData,mainFrame,this,width,height,tempCarP
arams,tempObstacleParams);
        simData.setSimulation(sim);

        if(jlayer != null)
            mainFrame.remove(jlayer);
        else
            mainFrame.remove(mainMenuPanel);
        jlayer = new JLayer<JPanel>(sim, layerUI);

        mainFrame.add(jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }

        simData.setRunning(true);
    }
    if(action.equals("Modify")) {
        if(jlayer != null)
            mainFrame.remove(jlayer);
        else
            mainFrame.remove(mainMenuPanel);
        jlayer = new JLayer<JPanel>(parametersMenu, layerUI);
        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
}

```

```

        }
        if(action.equals("Settings")) {
            if(jlayer != null)
                mainFrame.remove(jlayer);
            else
                mainFrame.remove(mainMenuPanel);
            jlayer = new JLayer<JPanel>(settingsMenu, layerUI);

            mainFrame.add (jlayer);
            mainFrame.revalidate();
            mainFrame.repaint();

            layerUI.start();
            if (!stopper.isRunning()) {
                stopper.start();
            }
        }
        if(action.equals("Back")) {
            if(jlayer != null)
                mainFrame.remove(jlayer);
            else
                mainFrame.remove(sim);
            jlayer = new JLayer<JPanel>(mainMenuPanel, layerUI);
            simData.setRunning(false);
            simData.setPaused(true);

            simData.setSimTime(0.0);
            try {
                sim.closeJoystick();
            } catch(Exception Nocontroller) {
                simData.logger.info("No Joystick to Close"); }

            sim.clearData();

            mainFrame.add (jlayer);
            mainFrame.revalidate();
            mainFrame.repaint();
            layerUI.start();
            if (!stopper.isRunning()) {
                stopper.start();
            }
        }
        if(action.equals("BackFromSettings")) {
            if(jlayer != null)
                mainFrame.remove(jlayer);
            jlayer = new JLayer<JPanel>(mainMenuPanel, layerUI);

```

```

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
    if(action.equals("BackFromParameters")) {
        if(jlayer != null)
            mainFrame.remove(jlayer);
        jlayer = new JLayer<JPanel>(mainMenuPanel, layerUI);

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
    if(action.equals("Quit")) {
        try {
            sim.close();
        } catch(Exception exp) {      simData.logger.info("Error
while closing simulation "+("+"+exp.getStackTrace()+""));
        }

        mainFrame.setVisible(false);
        mainFrame.dispose();

        System.exit(0);
    }
}
}
}

```

vdhils/HardwareInLoopSimulation.java:

```

/**
 *      \file HardwareInLoopSimulation.java
 *      \brief Hardware in loop Simulation Utilizing a EulerIntegration scheme.
 *
 *
 *      References:
 *

```

```

*      Revisions:
*          \li 2/6/2013 TFS
*          \li 11/19/2013 TFS
*          \li 4/23/2014 TFS
*          \li 8/15/2014 TFS
*
*      License:
*  This file is copyright 2013 by T Stevens and released under the Lesser GNU
*  Public License, version 2. It intended for educational use only, but its use
*  is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
*  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
*  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
*  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
*  TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
*  OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
*  OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****\todo Implement rewind, play, pause, feature for simulation
//\todo Makefile to erase and log files

```

```

package vdhils;
// Import Libraries
import vdhils.Vehicle.*;
import vdhils.GUI.*;
import vdhils.Graphics.*;
// GUI Imports
import javax.swing.*;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
import javax.swing.border.Border;

```

```

import javax.swing.filechooser.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.ArrayList;
import java.util.*;
import java.text.*;
// Joystick API Imports
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;
// File Operation Imports
import GeometryUtil.*;
import java.io.File;
import static java.nio.file.StandardOpenOption.*;
import java.nio.file.Files;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
// Error Handling Imports
import java.io.IOException;
// Serializable Import
import java.io.*;
import java.awt.Toolkit;

/**
 * \class HardwareInLoopSimulation
 * \brief Hardware in loop Simulation utilizes a simple Euler numerical integration
scheme to model car dynamics in a real-time simulated environment. The simulated
environments consists of
 *          other AI drivers and their vehicles & behaviors (traffic), roadways,
and road cones spaced evenly along the each side of each roadway. The user vehicle is
controlled through a usb
 *          gamecontroller featuring a force-feedback steering wheel and
throttle and braking floor pedals. In addition, the user vehicle simulates a LiDAR sensor
with adjustable range, resolution,
 *          and scan frequency in order to identify threatening obstacles within
the simulated environment and steer the vehicle according to path-planning algorithms
both of which run on an embedded
 *          development board (such as the BeagleBone or raspberry pi) . The
vehicle is mounted on a dynamometer.

```

```

*
*
    This class is responsible for instantiating all the roadways, obstacles, and
environment as well as user controls and data streams & communications to kick off the
simulation. Simulation starts
*
    in a PAUSED state, which may be toggled by pressing the space bar. This
class also serves as the base JPanel with a reference to its parent JFrame and
ActionListener to handle GUI manipulation.
*
*/
public class HardwareInLoopSimulation extends JPanel implements
ControllerEventListener, FeatureNotSupportedEventListener, ActionListener,
Serializable, Drawable, MouseListener, MouseMotionListener {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
serialized output stream */
    private static final long serialVersionUID = 7989078461104748962L;
    /** Specifies if this is the first time and indicates whether to create the simulation
update thread */
    private boolean firstEnable = true;
    /** Specifies the "PAUSED" font used during rendering */
    private Font pausedFont = new Font("Arail", Font.BOLD, 36);
    /** Update thread responsible for proper simulation timing and updating of
simulated vehicle dynamics and environment in real-time */
    private Thread updateThread;
    /** List of possible FFJoysticks */           /// \todo Clear this arraylist once
joystick is set
    private transient ArrayList<FFJoystick> joysticks;
    /** Specifies the user's FFJoystick that is used to drive the user's vehicle */
    private transient static FFJoystick joystick;
    /** Specifies the user's FFJoystick effect to realize the aligning moment at the
steering wheel caused by the car dynamics */
    private static Effect eff;
    /** Filechooser allowing user to specify a previously saved simulation in order to
load it */
    private final JFileChooser fc;
    /** Indicates whether the measure tool is active (via GUI) */
    private boolean measureToolEnable = false;
    /** Point's specified by user to be analyzed by the measure tool */
    private ArrayList<Point> clickPoints = new ArrayList<Point>();

    private static SimData simData;
    private final SimUpdater simUpdater;
    private Car car;
    private Roadway roadbounds;
    private Speedometer speedometer;
    private ObstacleAdmin obstacleAdmin;
}

```

```

    /** Button to exit application */
    private JButton exitButton;
    /** Holds reference to the main JFrame container */
    private static JFrame container;
    /** Button to switch back to main menu saving all logged data to allow for post-
processing */
    private JButton backButton;
    private JToolBar toolbar;
    private JButton toolBarExit;
    private JButton toolBarSave;
    private JButton toolBarLoad;
    private JButton toolBarBack;
    private JButton toolBarHelp;
    private JButton toolBarRecord;
    private JButton toolBarStop;
    private JButton toolBarPlay;
    private JButton toolBarMeasure;
    private int width;
    private int height;

    private Image cursorimage;
    private Toolkit toolkit;

    private byte DELIMITER = ' ';

    /**
     * Construct our sim and set it off running.
     */
    public HardwareInLoopSimulation(SimUpdater simUpdater, SimData simData,
final JFrame container, ActionListener MainListener, final int width, final int height) {
        // Get references to simulation data and update objects
        this.simUpdater = simUpdater;
        this.simData = simData;
        this.width = width;
        this.height = height;
        this.container = container;

        // if recordEnabled
        // create text files for streaming data output
        Date dateNow = new Date();
        // Create a unique & formatted date/time
        based filename
        SimpleDateFormat dateNowFormater = new
        SimpleDateFormat("yyMMddHHmmssZ");
        String formatDateNow = dateNowFormater.format(dateNow).toString();

```

```

String carRecordFP = "Car"+formatDateNow+".txt";
           // Car record stream filename
String obstacleRecordFP = "Obstacle"+formatDateNow+".txt";
           // Obstacle record stream filename
Path carPath = FileSystems.getDefault().getPath("Recorded
Data",carRecordFP);
BufferedWriter carRecord = null;
           // Car record output stream
BufferedWriter obstaclesRecords = null;
           // Obstacle record output stream
try {
    carRecord = new BufferedWriter(new
FileWriter(carPath.toFile().getAbsoluteFile()));      // Initialize Car record output stream
    simData.logger.info("Successfully created car data record file");
} catch( IOException x) {
    simData.logger.warning("Failed to create car data record file
("+x+"));
} catch(Exception otherExceptions) {
    simData.logger.warning("Failed to create car data record file
("+otherExceptions+"));
}
Path obstaclePath = FileSystems.getDefault().getPath("Recorded
Data",obstacleRecordFP);
try {
    obstaclesRecords = new BufferedWriter(new
FileWriter(obstaclePath.toFile().getAbsoluteFile()));      // Initialize Obstacle record
output stream
    simData.logger.info("Successfully created obstacle(s) data record
file");
} catch( IOException x) {
    simData.logger.warning("Failed to create obstacle(s) data record
file (" +x+"));
} catch(Exception otherExceptions) {
    simData.logger.warning("Failed to create obstacle(s) data record
file (" +otherExceptions+"));
}
// Create user car object
car =      new Car( new Dimension(width,height) , 0.2f , -80.0f ,
6.25f , 1.5708f , 21.0f , "/Resources/Textures/Cars/sportscar-jag.png" , false , carRecord );
// Create roadway object
roadbounds = new Roadway( new Dimension(width,height) );
/* Create GUI dashboard object */ /// \todo Different customizable
dashboard: speedometer, RRT indicator LED,
speedometer = new Speedometer( new Dimension(width,height),
"/Resources/Dashboard.png" , "/Resources/spedneedle2.png" );
// Create obstacle administrator object

```

```

obstacleAdmin = new ObstacleAdmin( new Dimension(width,height),
obstaclesRecords );

        setFocusable(true);
        addKeyBindings(this);
        // Initialize joystick
        initializeJoystick();
        // Create a file chooser
        fc = new JFileChooser();
        fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);

        createToolBar(MainListener);

        GridLayout holderLayout = new GridLayout(1, 0);
        holderLayout.setHgap(500);

        this.setLayout(new BorderLayout());
        this.add(toolbar, BorderLayout.NORTH);
        this.setOpaque(false);

        toolkit = Toolkit.getDefaultToolkit();
        cursorimage =
        toolkit.getImage("./Resources/Icons/mousemeasureicon.png");

        addMouseListener(this);
        addMouseMotionListener(this);
    }
    /**
     * Construct our sim and set it off running.
     */
    public HardwareInLoopSimulation(SimUpdater simUpdater, SimData simData,
final JFrame container, ActionListener MainListener, final int width, final int height,
final CarParameters carParams) {
        // Get references to simulation data and update objects
        this.simUpdater = simUpdater;
        this.simData = simData;
        this.width = width;
        this.height = height;
        this.container = container;

        // if recordEnabled
        // create text files for streaming data output
        Date dateNow = new Date();                                // Create a unique & formatted date/time
        based filename
    }
}

```

```

        SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
        String formatDateNow = dateNowFormater.format(dateNow).toString();
        String carRecordFP = "Car"+formatDateNow+".txt";
                // Car record stream filename
        String obstacleRecordFP = "Obstacle"+formatDateNow+".txt";
                // Obstacle record stream filename
        Path carPath = FileSystems.getDefault().getPath("Recorded
Data",carRecordFP);
        BufferedWriter carRecord = null;
                // Car record output stream
        BufferedWriter obstaclesRecords = null;
                // Obstacle record output stream
        try {
            carRecord = new BufferedWriter(new
FileWriter(carPath.toFile().getAbsoluteFile()));      // Initialize Car record output stream
            simData.logger.info("Successfully created car data record file");
        } catch( IOException x) {
            simData.logger.warning("Failed to create car data record file
("+x+"));
        } catch(Exception otherExceptions) {
            simData.logger.warning("Failed to create car data record file
("+otherExceptions+"));
        }
        Path obstaclePath = FileSystems.getDefault().getPath("Recorded
Data",obstacleRecordFP);
        try {
            obstaclesRecords = new BufferedWriter(new
FileWriter(obstaclePath.toFile().getAbsoluteFile()));      // Initialize Obstacle record
output stream
            simData.logger.info("Successfully created obstacle(s) data record
file");
        } catch( IOException x) {
            simData.logger.warning("Failed to create obstacle(s) data record
file ("+x+"));
        } catch(Exception otherExceptions) {
            simData.logger.warning("Failed to create obstacle(s) data record
file ("+otherExceptions+"));
        }
    }

    simData.scale = carParams.getScale();

    // Create user car object
    car =      new Car( new Dimension(width,height) , 0.2f, carParams,
false, carRecord );
    // Create roadway object

```

```

        roadbounds = new Roadway( new Dimension(width,height) );
        // Create GUI dashboard object
        /// \todo Different customizable dashboard: speedometer, RRT indicator
LED,
        speedometer = new Speedometer( new Dimension(width,height),
"/Resources/Dashboard.png", "/Resources/spedneedle2.png" );
        // Create obstacle administrator object
        obstacleAdmin = new ObstacleAdmin( new Dimension(width,height),
obstaclesRecords );
        if(carParams.getAlign())
            roadbounds.alignVehicle(car);

        setFocusable(true);
        addKeyBindings(this);
        // Initialize joystick
        initializeJoystick();
        // Create a file chooser
        fc = new JFileChooser();
        fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);

        createToolBar(MainListener);

        GridLayout holderLayout = new GridLayout(1, 0);
        holderLayout.setHgap(500);

        this.setLayout(new BorderLayout());
        this.add(toolbar, BorderLayout.NORTH);
        this.setOpaque(false);

        toolkit = Toolkit.getDefaultToolkit();
        cursorimage =
        toolkit.getImage("./Resources/Icons/mousemeasureicon.png");

        addMouseListener(this);
        addMouseMotionListener(this);
    }
/**/
 * Construct our sim and set it off running.
 */
public HardwareInLoopSimulation(SimUpdater simUpdater, SimData simData,
final JFrame container, ActionListener MainListener, final int width, final int height,
final CarParameters carParams, final ArrayList<CarParameters> obstacleParams) {
    // Get references to simulation data and update objects
    this.simUpdater = simUpdater;
    this.simData = simData;
    this.width = width;
}

```

```

        this.height = height;
        this.container = container;

        // if recordEnabled
        // create text files for streaming data output
        Date dateNow = new Date();                                // Create a unique & formatted date/time
based filename
        SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
        String formatDateNow = dateNowFormater.format(dateNow).toString();
        String carRecordFP = "Car"+formatDateNow+".txt";
                // Car record stream filename
        String obstacleRecordFP = "Obstacle"+formatDateNow+".txt";
                // Obstacle record stream filename
        Path carPath = FileSystems.getDefault().getPath("Recorded
Data",carRecordFP);
        BufferedWriter carRecord = null;
                // Car record output stream
        BufferedWriter obstaclesRecords = null;
                // Obstacle record output stream
try {
        carRecord = new BufferedWriter(new
FileWriter(carPath.toFile().getAbsoluteFile()));      // Initialize Car record output stream
        simData.logger.info("Successfully created car data record file");
    } catch(IOException x) {
        simData.logger.warning("Failed to create car data record file
(+x+)");
    } catch(Exception otherExceptions) {
        simData.logger.warning("Failed to create car data record file
(+otherExceptions+)");
    }
        Path obstaclePath = FileSystems.getDefault().getPath("Recorded
Data",obstacleRecordFP);
        try {
            obstaclesRecords = new BufferedWriter(new
FileWriter(obstaclePath.toFile().getAbsoluteFile()));      // Initialize Obstacle record
output stream
            simData.logger.info("Successfully created obstacle(s) data record
file");
        } catch(IOException x) {
            simData.logger.warning("Failed to create obstacle(s) data record
file (+x+)");
        } catch(Exception otherExceptions) {
            simData.logger.warning("Failed to create obstacle(s) data record
file (+otherExceptions+)");
        }
    }
}

```

```

    }

    simData.scale = carParams.getScale();

    // Create user car object
    car = new Car( new Dimension(width,height) , 0.2f, carParams,
false, carRecord );
    // Create roadway object
    roadbounds = new Roadway( new Dimension(width,height),
carParams.getRoadway(), carParams.getRoadWidth(), carParams.getConeSpacing(),
carParams.getConeSize() );
    // Create GUI dashboard object
    /// \todo Different customizable dashboard: speedometer, RRT indicator
LED,
    speedometer = new Speedometer( new Dimension(width,height),
"/Resources/Dashboard.png", "/Resources/spedneedle2.png" );
    // Create obstacle administrator object
    obstacleAdmin = new ObstacleAdmin( new Dimension(width,height),
0.2f, obstaclesRecords, obstacleParams );

    if(carParams.getAlign())
        roadbounds.alignVehicle(car);
    ArrayList<Car> tempCars = obstacleAdmin.getObstacles();
    for(int i=0; i<obstacleParams.size();i++) {
        if(obstacleParams.get(i).getAlign())
            roadbounds.alignVehicle(tempCars.get(i));
        if(obstacleParams.get(i).getEnableManuever())
            roadbounds.generateManuever(tempCars.get(i));
    }

    if(carParams.getEnableManuever()) {
        roadbounds.generateManuever(car);
    }

    setFocusable(true);
    addKeyBindings(this);
    // Initialize joystick
    initializeJoystick();
    // Create a file chooser
    fc = new JFileChooser();
    fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);

    createToolBar(MainListener);

    GridLayout holderLayout = new GridLayout(1, 0);
    holderLayout.setHgap(500);

```

```

        this.setLayout(new BorderLayout());
        this.add(toolbar, BorderLayout.NORTH);
        this.setOpaque(false);

        toolkit = Toolkit.getDefaultToolkit();
        cursorimage =
        toolkit.getImage("./Resources/Icons/mousemeasureicon.png");

        addMouseListener(this);
        addMouseMotionListener(this);
    }

    public void clearData() {
        obstacleAdmin.clearObstacles();
        roadbounds.clearMap();
    }

    private void createToolBar(ActionListener MainListener) {
        // Setting up the swing components
        toolbar = new JToolBar();
        toolbar.setFloatable(false);
        // Create toolbar icon's for each toolbar button
        ImageIcon iconExit = new ImageIcon("./Resources/Icons/ExitIcon.png");
        ImageIcon iconSave = new
        ImageIcon("./Resources/Icons/SaveIcon.png");
        ImageIcon iconLoad = new
        ImageIcon("./Resources/Icons/rsz_loadicon.png");
        ImageIcon iconBack = new
        ImageIcon("./Resources/Icons/rsz_backicon.png");
        ImageIcon iconHelp = new
        ImageIcon("./Resources/Icons/HelpIcon.png");
        ImageIcon iconRecord = new
        ImageIcon("./Resources/Icons/RecordIcon.png");
        ImageIcon iconStop = new ImageIcon("./Resources/Icons/StopIcon.png");
        ImageIcon iconPlay = new ImageIcon("./Resources/Icons/PlayIcon.png");
        ImageIcon iconMeasure = new
        ImageIcon("./Resources/Icons/measureicon.png");

        toolBarExit = new JButton(iconExit);
        toolBarSave = new JButton(iconSave);
        toolBarLoad = new JButton(iconLoad);
        toolBarBack = new JButton(iconBack);
        toolBarHelp = new JButton(iconHelp);
        toolBarRecord = new JButton(iconRecord);
        toolBarStop = new JButton(iconStop);
    }
}

```

```

toolBarPlay = new JButton(iconPlay);
toolBarMeasure = new JButton(iconMeasure);

toolBarExit.setFocusable(false);
toolBarSave.setFocusable(false);
toolBarLoad.setFocusable(false);
toolBarBack.setFocusable(false);
toolBarHelp.setFocusable(false);
toolBarRecord.setFocusable(false);
toolBarStop.setFocusable(false);
toolBarPlay.setFocusable(false);
toolBarMeasure.setFocusable(false);

toolBarRecord.setEnabled(false);
toolBarPlay.setEnabled(false);
toolBarStop.setEnabled(false);

toolBarExit.setToolTipText("Click this button to exit application");
toolBarSave.setToolTipText("Click this button to save this simulation");
toolBarLoad.setToolTipText("Click this button to load a previous
simulation");
toolBarBack.setToolTipText("Click this button to go back to the main
menu");
toolBarHelp.setToolTipText("Click this button to view the README and
project description file");
toolBarRecord.setToolTipText("Click this button to record the simulation
data to a delimited text file");
toolBarStop.setToolTipText("Click this button to stop/pause recording the
simulation data");
toolBarPlay.setToolTipText("Click this button to pause recording the
simulation data");
toolBarMeasure.setToolTipText("Click this button to measure distances
using the mouse to specify two points");

toolbar.add(toolBarSave);
toolbar.add(toolBarLoad);
toolbar.add(toolBarBack);
toolbar.addSeparator(new Dimension(32,32));
toolbar.add(toolBarRecord);
toolbar.add(toolBarStop);
toolbar.add(toolBarPlay);
toolbar.addSeparator(new Dimension(32,32));
toolbar.add(toolBarHelp);
toolbar.addSeparator(new Dimension(32,32));
toolbar.add(toolBarMeasure);
toolbar.addSeparator(new Dimension((int)(width - 15*32),32));

```

```

        toolbar.add(toolBarExit);

        toolBarBack.setActionCommand("Back");
        toolBarBack.addActionListener(MainListener);
        toolBarExit.setActionCommand("Quit");
        toolBarExit.setMnemonic('Q');
        toolBarExit.addActionListener(MainListener);
        toolBarSave.setActionCommand("Save");
        toolBarSave.addActionListener(this);
        toolBarLoad.setActionCommand("Load");
        toolBarLoad.addActionListener(this);
        toolBarPlay.setActionCommand("Play");
        toolBarPlay.addActionListener(this);
        toolBarMeasure.setActionCommand("Measure");
        toolBarMeasure.addActionListener(this);
        toolBarMeasure.setBackground(Color.red);

    }

    @Override
    public void mouseClicked(MouseEvent me) {

        if(measureToolEnable) {
            if(clickPoints.size()>=2) {
                measureToolEnable = false;
                toolBarMeasure.setBackground(Color.red);
                clickPoints.clear();
            } else {
                clickPoints.add(me.getPoint());
                Cursor c = toolkit.createCustomCursor(cursorimage,new
                Point(0,0), "img");
                setCursor(c);
            }
        } else {
            setCursor( Cursor.getDefaultCursor() );

            ArrayList<Car> tempCars = ObstacleAdmin.getObstacles();
            for (int i = 0; i < tempCars.size(); i++) {
                if (Geometry.isPointInsideRectangle(
tempCars.get(i).getBounds2D().getX()-tempCars.get(i).getBounds2D().getWidth()/2,
tempCars.get(i).getBounds2D().getY()-
tempCars.get(i).getBounds2D().getHeight()/2,
tempCars.get(i).getBounds2D().getX()+tempCars.get(i).getBounds2D().getWidth()
)/2,

```

```

        tempCars.get(i).getBounds2D().getY()+tempCars.get(i).getBounds2D().getHeight()
        ()/2,
    me.getX(), me.getY())) {//check if mouse is clicked within shape
        tempCars.get(i).togglePopup();
    }
}

if (Geometry.isPointInsideRectangle( car.getBounds2D().getX()-
car.getBounds2D().getWidth()/2,
car.getBounds2D().getY()-car.getBounds2D().getHeight()/2,
car.getBounds2D().getX()+car.getBounds2D().getWidth()/2,
car.getBounds2D().getY()+car.getBounds2D().getHeight()/2,
me.getX(), me.getY())) {//check if mouse is clicked within shape
    car.togglePopup();
}
}

@Override
public void mouseExited(MouseEvent e) {

}

@Override
public void mousePressed(MouseEvent e) {

}

@Override
public void mouseReleased(MouseEvent e) {

}

@Override
public void mouseEntered(MouseEvent e) {

}

public void mouseMoved(MouseEvent e) {

    if(Geometry.isPointInsideRectangle((int)speedometer.x_pos,(int)speedometer.y_p
os,(int)(speedometer.x_pos+speedometer.spriteWidth*speedometer.spriteScale),
(int)(speedometer.y_pos+speedometer.spriteHeight*speedometer.spriteScale),(int
)e.getX(),(int)e.getY())))

```

```

        { speedometer.mouseOverSpeedometer = true; }
    else { speedometer.mouseOverSpeedometer = false; }
}
public void mouseDragged(MouseEvent e) {

}

@Override
public void actionPerformed(ActionEvent evt) {
    String action = (String) evt.getActionCommand();
    if(simData.inputEnabled) {
        // Function Load simulation start
        simData.setPaused(true);
        if(action == "Load")
            loadSimulation();
        else if(action == "Save")
            saveSimulation();
    }
    if(action == "Measure") {
        if(measureToolEnable) {
            measureToolEnable = false;
            toolBarMeasure.setBackground(Color.red);
            clickPoints.clear();
        }
        else {
            measureToolEnable = true;
            toolBarMeasure.setBackground(Color.green);
        }
    }
}
public void closeRecordStreams() {
    car.closeRecordStream();
    ObstacleAdmin.closeRecordStreams();
}

public void close() {
    try {
        closeJoystick();
    } catch(Exception Nocontroller) { simData.logger.info("No Joystick to
Close"); }
    try {
        disconnect();
    } catch(Exception Noembeddedhardware) {
        simData.logger.info("No embedded hardware to disconnect from");
        closeRecordStreams();
    }
}
private void saveSimulation() {

```

```

simData.logger.info("Saving simulation data");
System.out.println("Saving...");

Date dateNow = new Date();
SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
String formatDateNow = dateNowFormater.format(dateNow).toString();
String rootSaveFP = "./Saves/VDHILS"+formatDateNow;

File saveFile = new File(rootSaveFP);
// Create the empty directory with default permissions, etc.
if(saveFile.mkdirs())
    simData.logger.info("Save file created successfully");
else
    simData.logger.warning("Save file creation failed exists");

boolean error = false;
try {
    car.writeCarToFile(rootSaveFP+"/Car");
} catch(Exception SaveError) {
    System.out.println("Error saving: "+SaveError);
    simData.logger.severe("Error saving car object: "+SaveError);
    error = true;
}

try {
    obstacleAdmin.writeObstaclesToFile(rootSaveFP+"/Obstacles");
} catch(Exception SaveError) {
    System.out.println("Error saving: "+SaveError);
    simData.logger.severe("Error saving obstacle objects:
"+SaveError);
    error = true;
}

try {
    roadbounds.writeRoadToFile(rootSaveFP+"/Road");
} catch(Exception SaveError) {
    System.out.println("Error saving: "+SaveError);
    simData.logger.severe("Error saving road object: "+SaveError);
    error = true;
}

try {
    simData.writeDataToFile(rootSaveFP+"/Data");
} catch(Exception SaveError) {
    System.out.println("Error saving: "+SaveError);
}

```

```

        simData.logger.severe("Error saving data object: "+SaveError);
        error = true;
    }

    try {

        speedometer.writeSpeedometerToFile(rootSaveFP+"/Speedometer");
    } catch(Exception SaveError) {
        System.out.println("Error saving: "+SaveError);
        simData.logger.severe("Error saving speedometer object:
"+SaveError);
        error = true;
    }

    JInternalFrame innerFrame = new JInternalFrame();
    add(innerFrame);
    innerFrame.setSize(300,150);
    innerFrame.pack();
    innerFrame.setVisible(true);
    if(!error)
        JOptionPane.showInternalMessageDialog(innerFrame,"Save
Completed Successfully.");
    else
        JOptionPane.showInternalMessageDialog(innerFrame,"Save
Failed.");
    remove(innerFrame);
    // Function save simulation end
}

private synchronized void loadSimulation() {
    simData.logger.info("Loading simulation data");
    System.out.println("Loading... ");

    final JFrame loadFrame = new JFrame("Load Simulation");
    loadFrame.setAlwaysOnTop(true);
    loadFrame.setSize(width,height);

    ActionListener fileListener = new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent ae) {
            //Process action performed
            String action = (String) ae.getActionCommand();
            System.out.println( action );
            if( action == "ApproveSelection" ) {

```

```

fc.getSelectedFile().getAbsolutePath();
from folder: "+folderToLoad );
from folder: "+folderToLoad);

String folderToLoad = (String)
System.out.println( "Loading simulation
simData.logger.info("Loading simulation

try {
    FileInputStream fileIn = new
    ObjectInputStream in = new
    car = (Car) in.readObject();
    in.close();
    fileIn.close();
    simData.logger.info("Loaded car
object successfully");

}catch(IOException i) {
    simData.logger.severe("Failed
loading car object: "+i.getStackTrace());
}

catch(ClassNotFoundException c) {
    System.out.println("Car class not
found");
    simData.logger.severe("Failed
loading car object, class not found: "+c.getStackTrace());
}

try {
    FileInputStream fileIn = new
    ObjectInputStream in = new
    obstacleAdmin = (ObstacleAdmin)
    in.readObject();
    in.close();
    fileIn.close();
    simData.logger.info("Loaded
obstacle objects successfully");

}catch(IOException i) {
    simData.logger.severe("Failed
loading obstacle objects: "+i.getStackTrace());
}

catch(ClassNotFoundException c) {
    System.out.println("ObstacleAdmin
class not found");
}

```

```

        simData.logger.severe("Failed
loading obtsacle objects, ObstacleAdmin class not found: "+c.getStackTrace());
    }

    try {
        FileInputStream fileIn = new
FileInputStream(folderToLoad+"/Road.ser");
        ObjectInputStream in = new
ObjectInputStream(fileIn);
        in.readObject();
        in.close();
        fileIn.close();
        simData.logger.info("Loaded road
object successfully");
    }catch(IOException i) {
        simData.logger.severe("Failed
loading road object: "+i.getStackTrace());
    }

    catch(ClassNotFoundException c) {
        System.out.println("Roadway class
not found");
        simData.logger.severe("Failed
loading road object, Roadway class not found: "+c.getStackTrace());
    }

    try {
        FileInputStream fileIn = new
FileInputStream(folderToLoad+"/Data.ser");
        ObjectInputStream in = new
ObjectInputStream(fileIn);
        in.readObject();
        in.close();
        fileIn.close();
        simData.info("Loaded data
object successfully");
    }catch(IOException i) {
        simData.logger.severe("Failed
loading data object: "+i.getStackTrace());
    }

    catch(ClassNotFoundException c) {
        System.out.println("SimData class
not found");
        simData.logger.severe("Failed
loading data object, SimData class not found: "+c.getStackTrace());
    }
}

```

```

        }

        try {
            FileInputStream fileIn = new
FileInputStream(folderToLoad+"/Speedometer.ser");
            ObjectInputStream in = new
ObjectInputStream(fileIn);
            speedometer = (Speedometer)
in.readObject();
            in.close();
            fileIn.close();
            simData.logger.info("Loaded
speedometer object successfully");
        }catch(IOException i) {
            simData.logger.severe("Failed
loading speedometer object: "+i.getStackTrace());
        }
        catch(ClassNotFoundException c) {
            System.out.println("Speedometer
class not found");
            simData.logger.severe("Failed
loading speedometer object, Speedometer class not found: "+c.getStackTrace());
        }

    } else
        System.out.println( "Cancel" );
        loadFrame.setVisible(false);
        container.setVisible(true);
        setVisible(true);
    }

};

fc.addActionListener( fileListener );
loadFrame.add( fc );
loadFrame.validate();

container.setVisible(false);
setVisible(false);
loadFrame.setVisible(true);
}

public synchronized void writeHILSToFile(String filepath) {
    try {
        // Serialize data object to a file
        ObjectOutputStream HILSOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
        HILSOut.writeObject(this);
    }
}

```

```

        HILSOut.close();

        //Serialize data object to a byte array
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        HILSOut = new ObjectOutputStream(bos);
        HILSOut.writeObject(this);
        HILSOut.close();
        byte[] buf = bos.toByteArray();
        SimData.logger.info("Saved HILS object successfully");
    } catch (IOException e) {
        SimData.logger.warning("Failed to write HILS object to file (" +
e.toString() + ")");
    }
}

public void addKeyBindings( JComponent jc ) {

    jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_SPACE, 0, false), "Space pressed");
    jc.getActionMap().put("Space pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            System.out.println("Space pressed");
            // if(simData.inputEnabled || !simData.getRunning()) {
            if(simData.getPaused()) {
                simData.setPaused(false);
                if(firstEnable) {
                    updateThread = new Thread(new
Runnable() {
            @Override
            public void run() {

                simUpdater.updateLoop();
            }
        });
    }
    updateThread.start();
}

        }
    else {
        simData.setPaused(true);
        updateThread.stop();
        // updateThread.interrupt();
    }
// }
}
});
```

```

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_UP, 0, false), "Up pressed");
    jc.getActionMap().put("Up pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(!simData.getJoystickControl() &&
simData.inputEnabled)
                car.keyPressed(KeyEvent.VK_UP);
        }
    });
};

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_DOWN, 0, false), "Down pressed");
    jc.getActionMap().put("Down pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(!simData.getJoystickControl() &&
simData.inputEnabled)
                car.keyPressed(KeyEvent.VK_DOWN);
        }
    });
};

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_RIGHT, 0, false), "Right pressed");
    jc.getActionMap().put("Right pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(!simData.getJoystickControl() &&
simData.inputEnabled)
                car.keyPressed(KeyEvent.VK_RIGHT);
        }
    });
};

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_LEFT, 0, false), "Left pressed");
    jc.getActionMap().put("Left pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(!simData.getJoystickControl() &&
simData.inputEnabled)
                car.keyPressed(KeyEvent.VK_LEFT);
        }
    });

```

```

        }
    });

    jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
        jc.getActionMap().put("Escape pressed", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                if(simData.getJoystickControl()) {
                    closeJoystick();
                    try {
                        simData.closeJoystick();
                    } catch(Exception nullJoystick) {
                        simData.logger.warning("No Joystick to Close"); }
                }
                System.exit(0);
            }
        });
}

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_PAGE_UP, 0, false), "Page up pressed");
    jc.getActionMap().put("Page up pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(simData.inputEnabled) {
                simData.scale += 0.5f;
                car.setScale(simData.scale/(simData.scale-0.5f));
                roadbounds.setScale(simData.scale,
car.getAbsCoords(), simData.scale-0.5f );
                obstacleAdmin.setScale( simData.scale );
            }
        }
    });
}

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_PAGE_DOWN, 0, false), "Page down pressed");
    jc.getActionMap().put("Page down pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            if(simData.inputEnabled) {
                simData.scale -= 0.5f;
                if(simData.scale >= 0.5f) {

```

```

        car.setScale(simData.scale/(simData.scale+0.5f));
                    roadbounds.setScale( simData.scale,
car.getAbsCoords(), simData.scale+0.5f );
                    obstacleAdmin.setScale( simData.scale );
                } else
                    simData.scale = 0.5f;
            }
        }
    });
}

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_A, 0, false), "A pressed");
jc.getActionMap().put("A pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        car.setOffsetX(car.getOffsetX()+10);

        roadbounds.updateOffsets(car.getOffsetX(),car.getOffsetY());

        obstacleAdmin.setOffsets(car.getOffsetX(),car.getOffsetY());
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_D, 0, false), "D pressed");
jc.getActionMap().put("D pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        car.setOffsetX(car.getOffsetX()-10);

        roadbounds.updateOffsets(car.getOffsetX(),car.getOffsetY());

        obstacleAdmin.setOffsets(car.getOffsetX(),car.getOffsetY());
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_W, 0, false), "W pressed");
jc.getActionMap().put("W pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        car.setOffsetY(car.getOffsetY()+10);
    }
});

```

```

        roadbounds.updateOffsets(car.getOffsetX(),car.getOffsetY());

        obstacleAdmin.setOffsets(car.getOffsetX(),car.getOffsetY());
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_S, 0, false), "S pressed");
jc.getActionMap().put("S pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        car.setOffsetY(car.getOffsetY()-10);

        roadbounds.updateOffsets(car.getOffsetX(),car.getOffsetY());

        obstacleAdmin.setOffsets(car.getOffsetX(),car.getOffsetY());
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_E, 0, false), "E pressed");
jc.getActionMap().put("E pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        ObstacleAdmin.enableObstaclesSC();
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_SPACE, 0, true), "Space released");
jc.getActionMap().put("Space released", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        System.out.println("Space released");
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_RIGHT, 0, true), "Right released");
jc.getActionMap().put("Right released", new AbstractAction() {
    @Override

```

```

        public void actionPerformed(ActionEvent ae) {
            if(!simData.getJoystickControl() &&
simData.inputEnabled)
                car.keyReleased(KeyEvent.VK_RIGHT);
        }
    });

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_LEFT, 0, true), "Left released");
jc.getActionMap().put("Left released", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(!simData.getJoystickControl() &&
simData.inputEnabled)
            car.keyReleased(KeyEvent.VK_LEFT);
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_UP, 0, true), "Up released");
jc.getActionMap().put("Up released", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(!simData.getJoystickControl() &&
simData.inputEnabled)
            car.keyReleased(KeyEvent.VK_UP);
    }
});

jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_DOWN, 0, true), "Down released");
jc.getActionMap().put("Down released", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        if(!simData.getJoystickControl() &&
simData.inputEnabled)
            car.keyReleased(KeyEvent.VK_DOWN);
    }
});

/// \todo R pressed/released to toggle recording
}

```

```

public void initializeJoystick() {
    simData.logger.info("Initializing JoystickManager...");
    System.out.println("Initializing JoystickManager...");
    try {
        JoystickManager.init();
        joysticks = JoystickManager.getAllFFJoysticks();
        joystick = JoystickManager.getFFJoystick();
    }
    catch (FFJoystickException e) {
        e.printStackTrace();
        simData.logger.warning("Joystick Initialization Error: " +
e.getErrorMessage());
    }
    if (joysticks.isEmpty()) {
        System.out.println("No Joysticks Detected"); // TODO No
joysticks detected resort to keyed input and enable user-input via keyboard
        simData.logger.info("No Joysticks Detected");
        simData.setJoystickControl(false);
    }
    else {
        simData.setJoystickControl(true);
        ControllerEventManager.addControllerEventListener(this);

        FeatureNotSupportedEventManager.addFeatureNotSupportedEventListener(this);

        setJoystick(joystick);
    }
}

@Override
public void controllerEventOccured(AdvancedControllerEvent event) {
    // we simply output it. but there are different classes of Events -
    // we could also distinguish between them and do more advanced stuff.
    //     System.out.println(event);
}

@Override
public void featureNotSupportedEventOccured(FeatureNotSupportedEvent event)
{
    System.out.println("Joystick feature not supported ("+event+ ")");
    simData.logger.warning("Joystick feature not supported +" + event + ")");
}

public void forwardCarThrottle(float acceleration) {
    if(simData.inputEnabled)
        car.setThrottle(acceleration);
}

```

```

        public void forwardBrakes(float brakes) {
            if(simData.inputEnabled)
                car.setBrakes(brakes);
        }
        public void forwardSteerangle(float steerangle) {
            if(simData.inputEnabled)
                car.setSteerangle(steerangle);
        }
        public void setJoystick(FFJoystick js) {
            joystick = js;
            if(joystick.isFFJoystick()) {
                simData.logger.info("Joystick set to: "+joystick.getName());
                System.out.println("Joystick set to: "+joystick.getName());
                if(joystick.getName().equals("Logitech Force 3D Pro
USB")||joystick.getName().equals("Logitech Driving Force GT USB")) {
                    // we set different dead zones for different axes
                    ///\todo allow user to specify dead zone
                    // joystick.setDeadZone(0, 0.01f);
                    // joystick.setDeadZone(1, 0.01f);
                    // joystick.setDeadZone(2, 0.2f);
                    // joystick.setDeadZone(3, 0.2f);
                }
                simData.setJoystickFF(true);
            }
            else {
                simData.logger.warning("-> ForceFeedback not supported");
                System.out.println("-> ForceFeedback not supported");
                simData.setJoystickFF(false);
            }
            simData.setJoystick(joystick);
        }
        public synchronized void closeJoystick() {
            simData.closeJoystick();
            JoystickManager.close();
        }
        public synchronized void disconnect() {
            car.disconnect();
        }
        public static void drawPixel(Graphics2D g, float x, float y, float size, Paint color)
    {
        Shape circle = new Ellipse2D.Float(x, y, size, size);
        g.setPaint(color);
        g.draw(circle);
        g.setPaint(color);
        g.fill(circle);
    }
}

```

```

public void draw(Graphics g) {
    car.setToolBarHeight(toolbar.getHeight()); // \todo fix this
    roadbounds.draw(g);

    obstacleAdmin.draw(g);
    car.draw(g);

    speedometer.draw(g);
    // collisions.draw(g); or drawCollisions(g);
    if(simData.getPaused()) {
        g.setColor(Color.red);
        g.setFont(pausedFont);
        g.drawString("PAUSED",width/2-
g.getFontMetrics().stringWidth("PAUSED")/2,height/2);
    }
    if(measureToolEnable) {
        for(int i = 0; i < clickPoints.size(); i+=2) {
            if(clickPoints.size()%2 == 0) {

                g.drawLine((int)clickPoints.get(i).getX(),(int)clickPoints.get(i).getY(),(int)clickP
oints.get(i+1).getX(),(int)clickPoints.get(i+1).getY());
                drawPixel((Graphics2D)g,
                (float)clickPoints.get(i+1).getX(), (float)clickPoints.get(i+1).getY(), 10.0f, Color.red);
                g.drawString("Distance:
"+clickPoints.get(i).distance(clickPoints.get(i+1))/SimData.scale+
(m)",(int)clickPoints.get(i+1).getX()+10,(int)clickPoints.get(i+1).getY()+10);
            }
            drawPixel((Graphics2D)g, (float)clickPoints.get(i).getX(),
            (float)clickPoints.get(i).getY(), 10.0f, Color.red);
        }
    }
    public synchronized void update(double elaspedTime) {

        // If recordObstacleData is true record obstacle data
        if(simData.recordObstacleData)
            obstacleAdmin.record(); //print obstacle number and description
        for each obstacle before printing data
        // Update manuever from last processed data to update next (resulting)
        time step
            car.updateSteerFromRRT(); // From last processed scan
            // Update physics for user and obstacle cars, the roadbounds, and the
            dashboard by advancing one time step
            car.update( elaspedTime );
        // If recordCarData is true record car data
        if(simData.recordCarData)
    }
}

```

```

        car.record();

        obstacleAdmin.update( elaspedTime, car.getOffsetX(), car.getOffsetY() );

        roadbounds.update( car.getAbsCoords(), car.getOffsetX(),
car.getOffsetY() );
        float carSpeed = car.getSpeed();

        speedometer.update( carSpeed, car.getThrottle(), car.getBrakes(),
carSpeed*elaspedTime*(1/3600.0) , car.getRRTIndicator() );
        speedometer.setSlideIndicator(car.isFrontSliding() || car.isRearSliding());

        /// \todo Only input objects and cones that are within radius of lidar scan
        /// \todo Turn cones into rectangle polygons for efficiency
        ArrayList<float[]> scanObstacles = new
ArrayList<float[]>(ObstacleAdmin.getAllEntityBounds());
        scanObstacles.addAll( roadbounds.getAllRoadCones() );
        // For next processed scan
        car.getLidar().setObstacles(scanObstacles);
        car.updateLiDAR( elaspedTime );
        // checkCollisions();
    }
    public boolean checkCollisions() {
        Rectangle2D.Float carBounds = car.getBounds2D();
        ArrayList<Car> obstacleCars = new ArrayList<Car>();
        obstacleCars = obstacleAdmin.getObstacles();
        for(int i = 0; i<obstacleCars.size(); i++) {
            Rectangle2D.Float obstacleBounds =
obstacleCars.get(i).getBounds2D();
        }
        return false;
    }
}

```

vdhils/ObstacleAdmin.java:

```

/**
 *
*   \file ObstacleAdmin.java
*
*
*   \brief ObstacleAdmin is a class to create all Obstacles and manage each obstacles
initial
*           placement and generate, artifical intelligence (AI) to follow road path,
running

```

```

*           the driver model of each obstacle vehicle, maintaining the car along the
specified
*           path (i.e. DLC, SLC, left, right, or steerangle(time) ), and performing and
maintaining
*           random car generation (depending on traffic conditions) and garbage
disposal (of car's
*           no longer apart of current simulation scene and that are not headed back
anytime soon.
*
*
*           References:
*
*           Revisions:
*           \li 4/24/2014 TFS
*
*           License:
*           This file is copyright 2014 by T Stevens and released under the Lesser GNU
*           Public License, version 2. It intended for educational use only, but its use
*           is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****  

*****

```

package vdhils;

import vdhils.Vehicle.\*;

```

import vdhils.Graphics.*;

import java.awt.*;
import java.net.URL;
import java.util.Random;
import java.util.ArrayList;
import java.awt.image.*;
import GeometryUtil.*;
import java.awt.geom.*;
import java.io.*;
import static java.lang.Math.*;

/**
 *      \class ObstacleAdmin
 *      \brief ObstacleAdmin class handles all the obstacle vehicles updates, movements,
steering manuevers, etc...
 */
public class ObstacleAdmin implements Serializable, Drawable, Recordable {
/// \bug Obstacle number not printing to record file
    /** Serializable UID for saving/loadding functionality */
    private static final long serialVersionUID = 7989078461104748967L;
    /** Obstacle array holding all other vehicles in simulation other than player/user
*/
    private static ArrayList<Car> Obstacles = new ArrayList<Car>();
    private BufferedWriter recordStream;
    private Dimension screen;
    /// \todo Randomly generate obstacle cars and locations
    public ObstacleAdmin(Dimension screen , BufferedWriter recordStream) {
        this.recordStream = recordStream;
        this.screen = screen;
    }
    /** Overloaded constructor handling obstacle parameters
     * \param screen           Screen dimensions
     * \param recordStream      Output record stream for recorded obstacle
data
     * \param obstacleParams   Obstacle vehicle parameters
    */
    public ObstacleAdmin(Dimension screen , float spritescale, BufferedWriter
recordStream, ArrayList<CarParameters> obstacleParams) {
        this(screen, recordStream);
        for(int i = 0; i<obstacleParams.size(); i++) {
            CarParameters tempParams = obstacleParams.get(i);
            Obstacles.add( new Car( screen, spritescale, tempParams, true,
recordStream ) );
        }
    }
}

```

```

    /**
     * This function writes all the obstacle vehicles data to the record file data
     * stream
     */
    public void record() {
        for(int i = 0; i < Obstacles.size(); i++) {
            Car temp = Obstacles.get(i);
            try {
                recordStream.write("Obstacle #: ");
                recordStream.write(i);
                recordStream.newLine();
            } catch(Exception e) { SimData.logger.warning("Failed to write to
obstacle record stream ("+e.getStackTrace()+"")"); }
                temp.record();
            }
        }
        public static void closeRecordStreams() {
            for(int i = 0; i < Obstacles.size(); i++) {
                Car temp = Obstacles.get(i);
                temp.closeRecordStream();
            }
        }
        public synchronized void update(double elaspedTime) {
            for(int i = 0; i < Obstacles.size(); i++) {
                Car temp = Obstacles.get(i);
                temp.update( elaspedTime );
            }
        }
        public synchronized void update(double elaspedTime, float offsetX, float offsetY)
{
            for(int i = 0; i < Obstacles.size(); i++) {
                Car temp = Obstacles.get(i);
                temp.update( elaspedTime );
            }
            setOffsets(offsetX,offsetY);
        }
        public void setOffsets(float offsetX, float offsetY) {
            for(int i = 0; i < Obstacles.size(); i++) {
                Car temp = Obstacles.get(i);
                temp.setOffsetX(offsetX);
                temp.setOffsetY(offsetY);
            }
        }
        public void draw(Graphics g) {

```

```

        for(int i = 0; i < Obstacles.size(); i++) {
            Car temp = Obstacles.get(i);
            temp.draw(g);
        }
    }

    public static void enableObstaclesSC() {
        for(int i = 0; i < Obstacles.size(); i++) {
            Car temp = Obstacles.get(i);
            boolean toggleEnable = !temp.speedController.getEnable();
            System.out.println("TOGGLE: "+toggleEnable);
            temp.enableSpeedControl(toggleEnable);
        }
    }

    public static ArrayList<Car> getObstacles() {
        return Obstacles;
    }

    public void clearObstacles() {
        Obstacles.clear();
    }

    public synchronized void writeObstaclesToFile(String filepath) {
        try {
            // Serialize data object to a file
            ObjectOutputStream ObstaclesOut = new
ObjectOutputStream(new FileOutputStream(filepath+".ser"));
            ObstaclesOut.writeObject(this);
            ObstaclesOut.close();

            //Serialize data object to a byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            ObstaclesOut = new ObjectOutputStream(bos);
            ObstaclesOut.writeObject(this);
            ObstaclesOut.close();
            byte[] buf = bos.toByteArray();
            SimData.logger.info("Obstacles saved successfully");
        } catch (IOException e) {
            SimData.logger.warning("Failed to write obstacle objects to file ("+
e.toString() + ")");
        }
    }

    public void setScale( float scale ) {
        for(int i = 0; i < Obstacles.size(); i++) {
            Car temp = Obstacles.get(i);
            temp.setScale( scale );
        }
    }
}

```

```

        }
    }

    public static ArrayList<float[]> getAllEntityBounds() {
        ArrayList<float[]> returnObstacles = new ArrayList<float[]>();
        for(int i = 0; i < Obstacles.size(); i++) {
            returnObstacles.addAll( Obstacles.get(i).getEntityBounds() );
        }
        return returnObstacles;
    }
}

```

vdhils/Roadway.java:

```

/**
 *      \file Roadway.java
 *
 *      \brief Roadway class responsible for using graphic context to draw road tiles. In
 *              addition this class initializes, maintains, and generates new tiles randomly
 *              using roadrules to create seemingly infinite maps, unless of course
 *              generated
 *              tiles form a closed path.
 *
 *      \author Thomas Stevens
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.*/

```

```

//*****
*****



package vdhils;

/// \todo Clean up imports
// Import java API's/libraries
import vdhils.Graphics.*;
import vdhils.Vehicle.Car;
// GUI Imports
import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Random;
import static java.lang.Math.*;
import java.net.URL;
import java.util.ArrayList;
import java.io.*;
import GeometryUtil.*;
/**  \class Roadway
 *   \brief Roadway class responsible for using graphic context to draw the map. The
map
 *           is composed of connected road tiles that form various roadway
configurations. In
 *           addition this class initializes, maintains, and generates new tiles
dynamically
 *           using the roadrules to create seemingly infinite maps.
*/
public class Roadway implements Serializable {
    /** Serializable UID for saving/loading functionality */
    private static final long serialVersionUID = 7989078461104748966L;
    /** Keeps track of map origin offset */
    private float offsetX = 0.0f;
    private float offsetY = 0.0f;
    /** Road cone spacing in meters */
    private float spacing = 1.22f;
    /** Road cone size in pixels */
    private float size = 5f;
    /** Iterations for horizontal road section's road cone generation */
    private int iterations_width;
    /** Iterations for vertical road section's road cone generation */
    private int iterations_height;
    /** Car x-position index in map coords (index representing current map tile) */
    private int curMapX = 0;
}

```

```

/** Car y-position index in map coords (index representing current map tile) */
private int curMapY = 0;
private int minXindex = 0;
private int minYindex = 0;
private int maxXindex = 0;
private int maxYindex = 0;
/** Road Scale relative to Simulation Scale in order for roadScale*simData.scale
= */
private static float roadScale = 0.15f;
/** Road Width as a percentage of the tile width (pixels) */
private final float roadWidthAsPercentOfTileWidth = 0.25f;
/** Road With as a percentage of the tile height (pixels) */
private final float roadWidthAsPercentOfTileHeight = 0.325f;
/** The number of road tiles that compose the simulation map */
private int numTiles;
protected float change;
/** The road bounds detected by the user's vehicle */
private ArrayList<float[]> RoadBounds = new ArrayList<float[]>();
/** The simulation road tile map */
private ArrayList<RoadTile> Map = new ArrayList<RoadTile>();
/// \todo Implement dirt roadway sections with variable coefficients of friction
private final String fourWayIntersection =
"/Resources/Road_Tiles/4wayroadpiece350x262_Textured.png";
private final String fourWay_HorizRitAway =
"/Resources/Road_Tiles/crosshorizroadpiece350x262_Textured.png";
private final String fourWay_VertRitAway =
"/Resources/Road_Tiles/crossvertroadpiece350x262_Textured.png";
private final String Tee_HorizRitAway =
"/Resources/Road_Tiles/Tsectionhorizroadpiece350x262_Textured.png";
private final String Tee_VertRitAway =
"/Resources/Road_Tiles/Tsectionvertroadpiece350x262_Textured.png";
private final String Tee_HorizUpRitAway =
"/Resources/Road_Tiles/Tsectionhorizuproadpiece350x262_Textured.png";
private final String Tee_VertRightRitAway =
"/Resources/Road_Tiles/Tsectionrightvertroadpiece350x262_Textured.png";
private final String LeftCurveRoad =
"/Resources/Road_Tiles/curveleftroadpiece350x262_Textured.png";
private final String RightCurveRoad =
"/Resources/Road_Tiles/curverightroadpiece350x262_Textured.png";
private final String LeftUpCurveRoad =
"/Resources/Road_Tiles/curveleftuproadpiece350x262_Textured.png";
private final String RightUpCurveRoad =
"/Resources/Road_Tiles/curverightuproadpiece350x262_Textured.png";
private final String HorizStraightRoad =
"/Resources/Road_Tiles/horizstraightroadpiece350x262_Textured.png";

```

```

        private final String VertStraightRoad =
"/Resources/Road_Tiles/vertstraightroadpiece350x262_Textured.png";
        private final String GrassRoad =
"/Resources/Road_Tiles/grasspiece350x262.png";

protected static int tileWidth;
protected static int tileHeight;

private Dimension screen;
private String roadType;

public Roadway(Dimension screen) {
    this.screen = screen;
    RoadTile tempTile = new
RoadTile(screen,roadScale*SimData.scale,fourWayIntersection,0,0,false);
    tileWidth = tempTile.spriteWidth;
    tileHeight = tempTile.spriteHeight;
    iterations_width =
(int)((tileWidth*tempTile.spriteScale)/(spacing*SimData.scale));
    iterations_height =
(int)((tileHeight*tempTile.spriteScale)/(spacing*SimData.scale));
}

public Roadway(Dimension screen, String roadType) {
    this(screen);
    this.roadType = roadType;

    initializeMap();
}

public Roadway(Dimension screen, String roadType, float desiredRoadWidth) {
// desired road width in meters
    this.screen = screen;
    this.roadType = roadType;
    roadScale = desiredRoadWidth/86.325f;
    RoadTile tempTile = new
RoadTile(screen,roadScale*SimData.scale,fourWayIntersection,0,0,false);
    tileWidth = tempTile.spriteWidth;
    tileHeight = tempTile.spriteHeight;
    iterations_width =
(int)((tileWidth*tempTile.spriteScale)/(spacing*SimData.scale));
    iterations_height =
(int)((tileHeight*tempTile.spriteScale)/(spacing*SimData.scale));

    initializeMap();
}

```

```

        public Roadway(Dimension screen, String roadType, float desiredRoadWidth,
float coneSpacing, float coneSize) { // desired road width in meters
            this.screen = screen;
            this.spacing = coneSpacing;
            this.size = coneSize*SimData.scale;
            this.roadType = roadType;
            roadScale = desiredRoadWidth/86.325f;
            RoadTile tempTile = new
RoadTile(screen,roadScale*SimData.scale,fourWayIntersection,0,0,false);
            tileWidth = tempTile.spriteWidth;
            tileHeight = tempTile.spriteHeight;
            iterations_width =
(int)((tileWidth*tempTile.spriteScale)/(spacing*SimData.scale));
            iterations_height =
(int)((tileHeight*tempTile.spriteScale)/(spacing*SimData.scale));

            initializeMap();
        }

        public ArrayList<float[]> getRoadBounds() {
            return RoadBounds;
        }

        private void refreshMap() {
            while( -offsetX < minXindex*tileWidth*roadScale*SimData.scale ) {
                // add column of tiles
                for(int i = minYindex; i < (maxYindex - minYindex+1); i++) {
                    int[][] surrIndices = new int[][] { { minYindex-1, i-1 }, { minYindex-1, i+1 }, { minYindex-2, i }, { minYindex, i } };
                    boolean success = addNewTile( minYindex-1, i,
surrIndices );
                    }
                    minYindex--;
                }
                while( -offsetX+screen.getWidth() >
(maxXindex*tileWidth+tileWidth)*roadScale*SimData.scale ) {
                    // add column of tiles
                    for(int i = minYindex; i < (maxYindex - minYindex+1); i++) {
                        int[][] surrIndices = new int[][] { { maxXindex+1, i-1 }, { maxXindex+1, i+1 }, { maxXindex, i }, { maxXindex+2, i } };
                        boolean success = addNewTile( maxXindex+1, i,
surrIndices );
                    }
                    maxXindex++;
                }
            }

```

```

        while( -offsetY < minYindex*tileHeight*roadScale*SimData.scale ) {
            // add row of tiles
            for(int i = minXindex; i < (maxXindex - minXindex+1); i++) {
                int[][] surrIndices = new int[][] { { i, minYindex-2 }, { i,
minYindex }, { i-1, minYindex-1 }, { i+1, minYindex-1 } };
                boolean success = addNewTile( i, minYindex-1,
surrIndices );
            }
            minYindex--;
        }
        while( -offsetY+screen.getHeight() >
(maxYindex*tileHeight+tileHeight)*roadScale*SimData.scale ) {
            // add row of tiles
            for(int i = minXindex; i < (maxXindex - minXindex + 1); i++) {
                int[][] surrIndices = new int[][] { { i, maxYindex }, { i,
maxYindex+2 }, { i-1, maxYindex+1 }, { i+1, maxYindex+1 } };
                boolean success = addNewTile( i, maxYindex+1,
surrIndices );
            }
            maxYindex++;
        }
        // remove offscreen roadTiles
        removeOffscreenTiles();
    }

    public void updateOffsets(float offsetX, float offsetY) {
        this.offsetX = offsetX;
        this.offsetY = offsetY;

        refreshMap();
    }

    public void removeOffscreenTiles() {
        int minoffsetXindex = (int)((-
offsetX)/(tileWidth*roadScale*SimData.scale));
        int maxoffsetXindex = (int)((-
offsetX+screen.getWidth())/(tileWidth*roadScale*SimData.scale));
        int minoffsetYindex = (int)((-
offsetY)/(tileHeight*roadScale*SimData.scale));
        int maxoffsetYindex = (int)((-
offsetY+screen.getHeight())/(tileHeight*roadScale*SimData.scale));
        for(int i=0; i < Map.size(); i++) {
            RoadTile roadTile = Map.get(i);

            if((roadTile.mapX>(maxoffsetXindex+1))||(roadTile.mapX<(minoffsetXindex-
1))) {

```

```

        Map.remove(i);
        continue;
    }

    if((roadTile.mapY>(maxoffsetYindex+1))||(roadTile.mapY<(minoffsetYindex-1))) {
        Map.remove(i);
        continue;
    }
}

public float getRoadWidth() {
    return (roadWidthAsPercentOfTileWidth*tileWidth);
}

public float getRoadHeight() {
    return (roadWidthAsPercentOfTileHeight*tileHeight);
}

public void draw(Graphics g) {
    for(int i = 0; i < Map.size(); i++) {
        RoadTile tempTile = Map.get(i);
        tempTile.draw(g);
    }
}

public synchronized void writeRoadToFile(String filepath) {
    try {
        // Serialize data object to a file
        ObjectOutputStream RoadOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
        RoadOut.writeObject(this);
        RoadOut.close();

        //Serialize data object to a byte array
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        RoadOut = new ObjectOutputStream(bos);
        RoadOut.writeObject(this);
        RoadOut.close();
        byte[] buf = bos.toByteArray();
        SimData.logger.info("Save road successfully");
    } catch (IOException e) {
        SimData.logger.warning("Failed to write road object to file (" +
e.toString() + ")");
    }
}

```

```

    }

    private void initializeMap() {
        // Check surrounding location's that appear on screen for tiles
        int numTilesNeeded = (int) ceil(
            (screen.getWidth()*screen.getHeight())/(tileWidth*tileHeight) );
        int maxMapX = (int)ceil(
            (screen.getWidth())/(tileWidth*roadScale*SimData.scale));
        int maxMapY = (int)ceil(
            (screen.getHeight())/(tileHeight*roadScale*SimData.scale));
        for(int i=0; i < maxMapX; i++) {
            for(int j=0; j < maxMapY; j++) {
                int[][] surrIndices = new int[][] { { i, j-1 }, { i, j+1 }, { i-1,
                    j }, { i+1, j } };
                boolean success = addNewTile( i, j, surrIndices );
                if(success) {
                    if( i < minXIndex)
                        minXIndex = i;
                    if( i > maxXIndex)
                        maxXIndex = i;
                    if( j < minYIndex)
                        minYIndex = j;
                    if(j > maxYIndex)
                        maxYIndex = j;
                }
            }
        }
    }

    public void clearMap() {
        Map.clear();
    }

    private void drawGeneralPath(Graphics2D g2, float[] Points) {
        GeneralPath polygon = new
        GeneralPath(GeneralPath.WIND_EVEN_ODD,Points.length);
        for(int i=0; i < Points.length;) {
            if (i == 0)
                polygon.moveTo(Points[i], Points[i+1]);
            polygon.lineTo(Points[i], Points[i+1]);
            i=i+2;
        }
        polygon.closePath();
        // if(color == 1)
        g2.setColor(Color.ORANGE);
        // else if(color == 2)
    }
}

```

```

        // g2.setColor(obstacleFill);
// else if (color == 3)
    // g2.setColor(coneFill);
// else if(color ==4)
    // g2.setColor(arrowColor);
// else
    // g2.setColor(Color.BLACK);
// if(fillpatt == 1)
    // g2.fill(polygon);
// else {
    g2.fill(polygon);
    // g2.setColor(Color.RED);
    // g2.draw(polygon);
}
}

private String getTileFilePath(int fileIndex) {
    String retString = "";
    switch(fileIndex) {
        case 0:
            retString = fourWayIntersection;
            break;
        case 1:
            retString = HorizStraightRoad;
            break;
        case 2:
            retString = VertStraightRoad;
            break;
        case 3:
            retString = fourWay_HorizRitAway;
            break;
        case 4:
            retString = fourWay_VertRitAway;
            break;
        case 5:
            retString = GrassRoad;
            break;
        case 6:
            retString = Tee_HorizRitAway;
            break;
        case 7:
            retString = Tee_VertRitAway;
            break;
        case 12:
            retString = Tee_HorizUpRitAway;
            break;
    }
}

```

```

        case 13:
            retString = Tee_VertRightRitAway;
            break;
        case 8:
            retString = LeftCurveRoad;
            break;
        case 9:
            retString = RightCurveRoad;
            break;
        case 10:
            retString = LeftUpCurveRoad;
            break;
        case 11:
            retString = RightUpCurveRoad;
            break;
    }
    return retString;
}

private boolean addNewTile(int tileMapX, int tileMapY, int[][] surrIndices) {
    Random random = new Random( System.currentTimeMillis() );
    boolean addTileFail = false;
    int randIndex=0;

    do {
        randIndex = random.nextInt(14);
        if(randIndex == 5) {
            randIndex = random.nextInt(14);
        }
        try {
            switch(roadType.toLowerCase()) {
                case "horizontal":
                    randIndex = 1;
                    break;
                case "vertical":
                    randIndex = 2;
                    break;
                case "oval":
                    while((randIndex !=1)&&(randIndex !=2)&&(randIndex
                    !=8)&&(randIndex != 9)&&(randIndex != 10)&&(randIndex != 11)) {
                        randIndex = random.nextInt(11)+1;
                    }
                    break;
                case "none":
                    randIndex = 5;
                    break;
            }
        }
    }
}

```

```

        case "closed oval":
            while((randIndex !=8)&&(randIndex != 9)&&(randIndex
!= 10)&&(randIndex != 11)&&(randIndex != 5)) {
                randIndex = random.nextInt(14);
            }
            break;
        case "no curves":
            randIndex = random.nextInt(2)+3;
            break;
        // default:
        // randIndex = random.nextInt(14);
        // if(randIndex == 5) {
            // randIndex = random.nextInt(14);
        // }
        // break;
    }
} catch(Exception exp) {}
RoadTile randTile = new RoadTile( screen, roadScale*SimData.scale,
getTileFilePath(randIndex), tileMapX, tileMapY, false );

boolean roadRulesCheck = true;
// Check road rules for (randomly) generated tile to see if adding to map is
appropriate
for(int i = 0; i < 4; i++) {
    RoadTile checkTile = null;
    boolean match = false;
    for(int j = 0; j < Map.size(); j++) {
        checkTile = Map.get(j);
        if( (checkTile.mapX ==
surrIndices[i][0])&&(checkTile.mapY == surrIndices[i][1])) {
            match = true;
            break;
        }
    }
    if(match == true) {
        switch(i) {
            case 0: // TOP/ABOVE
                if( randTile.RoadRules[1] !=
checkTile.RoadRules[2])
                    roadRulesCheck = false;
                break;
            case 1: // Bottom/BELOW
                if( randTile.RoadRules[2] !=
checkTile.RoadRules[1])
                    roadRulesCheck = false;
                break;
        }
    }
}

```

```

        case 2: // LEFT
            if( randTile.RoadRules[3] != checkTile.RoadRules[4])
                roadRulesCheck = false;
            break;
        case 3: // RIGHT
            if( randTile.RoadRules[4] != checkTile.RoadRules[3])
                roadRulesCheck = false;
            break;
        }
    }
    if( roadRulesCheck == false )
        break;
}
// If road rules check or tile is blank (grass), add it to the map
if((roadRulesCheck)|(randTile.tileID == 5)) {
    Map.add(randTile);
    switch(randTile.tileID) {
        case 0:
            //Create Road Cones Dynamically
            for(int i = 0; i < iterations_width; i++) {
                if((i == iterations_width/2)|(i ==
(int)(iterations_width/2)+1))
                    continue;
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );

```

```

        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale ) );
}
for(int i = 0; i < iterations_height; i++) {
    if((i == iterations_height/2)|| (i ==
(int)(iterations_height/2)+1))
        continue;
    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

```

```

screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) ));
                    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile
.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
                }
            break;
        case 1:
            //Create Road Cones Dynamically
            for(int i = 0; i < iterations_width; i++) {
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil

```

```

e.spriteScale-screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale ) );
            }
        break;
    case 2:
        //Create Road Cones Dynamically
        for(int i = 0; i < iterations_height; i++) {
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
                (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
                (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

```

```

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,

```

(float)(randTile.x\_pos+(tileWidth/2)\*randTile.spriteScale+(getRoadHeight()/2)\*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y\_pos-
screen.getHeight()/2+spacing\*SimData.scale\*i+size/2)/SimData.scale,

(float)(randTile.x\_pos+(tileWidth/2)\*randTile.spriteScale+(getRoadHeight()/2)\*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y\_pos-
screen.getHeight()/2+spacing\*SimData.scale\*i+size/2)/SimData.scale,

(float)(randTile.x\_pos+(tileWidth/2)\*randTile.spriteScale+(getRoadHeight()/2)\*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y\_pos-
screen.getHeight()/2+spacing\*SimData.scale\*i-size/2)/SimData.scale ) );
}

break;

case 3:

//Create Road Cones Dynamically

for(int i = 0; i < iterations\_width; i++) {

if((i == iterations\_width/2)|| (i ==
(int)(iterations\_width/2)+1))

continue;

randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x\_pos+spacing\*SimData.scale\*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y\_pos+(tileHeight/2)\*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)\*randTile.spriteScale-size/2)/SimData.scale,

(float)(randTile.x\_pos+spacing\*SimData.scale\*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y\_pos+(tileHeight/2)\*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)\*randTile.spriteScale+size/2)/SimData.scale,

(float)(randTile.x\_pos+spacing\*SimData.scale\*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y\_pos+(tileHeight/2)\*randTil

```

e.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
                                         randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale ) );
                                         }
                                         for(int i = 0; i < iterations_height; i++) {
                                         if((i == iterations_height/2)|| (i ==
(int)(iterations_height/2)+1))
                                         continue;
                                         randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
                                         (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-

```

```

        (getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
                    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale ) );
                }
            break;
        case 4:
            //Create Road Cones Dynamically
            for(int i = 0; i < iterations_width; i++) {
                if((i == iterations_width/2)||(i ==
(int)(iterations_width/2)+1))
                    continue;
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
  

        (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-

```

```

size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) ));
}
for(int i = 0; i < iterations_height; i++) {
    if((i == iterations_height/2)|| (i ==
(int)(iterations_height/2)+1))
        continue;
    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-

```

```

size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
                    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale ) );

        }
        break;
        case 6:
            //Create Road Cones Dynamically
            for(int i = 0; i < iterations_width; i++) {
                randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-

```

```

screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
                                         if((i == iterations_width/2)|| (i ==
(int)(iterations_width/2)+1))
                                         continue;
                                         randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                                         (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale ) );
                                         }
                                         for(int i = 0; i < iterations_height; i++) {
                                         if(i <= (int)(iterations_height/2)+1)
                                         continue;
                                         randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,

```

```

        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale );
        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
        (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
        break;
    case 7:
        //Create Road Cones Dynamically
        for(int i = 0; i < iterations_width; i++) {
            if(i >= (int)(iterations_width/2)-1)
                continue;

```

```

        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
    }
    for(int i = 0; i < iterations_height; i++) {
        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile

```

```

.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,

(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
if(i == iterations_height/2)||(i ==
(int)(iterations_height/2)+1))
continue;
randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
}
break;
case 12:

```

```

        //Create Road Cones Dynamically
        for(int i = 0; i < iterations_width; i++) {
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale ) );
            if((i == iterations_width/2)|| (i ==
(int)(iterations_width/2)+1))
                continue;
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-
screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
                (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTil-
e.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale ) );
        }
    }
}

```

```

        for(int i = 0; i < iterations_height; i++) {
            if(i >= (int)(iterations_height/2))
                continue;
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale );
        );
    }
}

```

```

        }
    break;
    case 13:
        //Create Road Cones Dynamically
        for(int i = 0; i < iterations_width; i++) {
            if(i <= (int)(iterations_width/2)+1)
                continue;
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*randTile.spriteScale+size/2)/SimData.scale,
(float)(randTile.x_pos+spacing*SimData.scale*i-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos+(tileHeight/2)*randTile.spriteScale-

```

```

e.spriteScale-screen.getHeight()/2+(getRoadWidth()/2)*randTile.spriteScale-
size/2)/SimData.scale ) );
}
for(int i = 0; i < iterations_height; i++) {
    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale.getgetWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale-
(getRoadHeight()/2)*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale ) );
if((i == iterations_height/2)||i ==
(int)(iterations_height/2)+1))
    continue;
    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle(
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*randTile
.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i-size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,
(float)(randTile.x_pos+(tileWidth/2)*randTile.spriteScale+(getRoadHeight()/2)*r
andTile.spriteScale-screen.getWidth()/2+size/2)/SimData.scale,(float)(randTile.y_pos-
screen.getHeight()/2+spacing*SimData.scale*i+size/2)/SimData.scale,

```



```

t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

    (float)(randTile.y_pos+(262.3*pow(1-t,3)+184.1*3*t*pow(1-t,2)+171.3*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
}
// outside curve
iterationsAlongBezier =
(int)((313.9*randTile.spriteScale)/SimData.scale)/spacing);
for(int i = 0; i < iterationsAlongBezier; i++) {
    double t =
((float)(i)/(float)(iterationsAlongBezier-1));
    randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(131.8*pow(1-t,3)+139.2*3*t*pow(1-
t,2)+253.3*3*(1-t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
  

    (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
  

    (float)(randTile.x_pos+(131.8*pow(1-t,3)+139.2*3*t*pow(1-t,2)+253.3*3*(1-
t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

    (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,
  

    (float)(randTile.x_pos+(131.8*pow(1-t,3)+139.2*3*t*pow(1-t,2)+253.3*3*(1-

```

```

t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
    }
    break;
    case 9:
        iterationsAlongBezier =
(int)((186.9*randTile.spriteScale)/SimData.scale)/spacing);
        // inside curve
        for(int i = 0; i < iterationsAlongBezier; i++) {
            double t =
((float)(i)/(float)(iterationsAlongBezier-1));
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(133.8*pow(1-t,3)+130.7*3*t*pow(1-
t,2)+56.4*3*(1-t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(262.3*pow(1-t,3)+184.1*3*t*pow(1-t,2)+171.3*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(133.8*pow(1-t,3)+130.7*3*t*pow(1-t,2)+56.4*3*(1-
t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(262.3*pow(1-t,3)+184.1*3*t*pow(1-t,2)+171.3*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,

```

```

        (float)(randTile.x_pos+(133.8*pow(1-t,3)+130.7*3*t*pow(1-t,2)+56.4*3*(1-
t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
        (float)(randTile.y_pos+(262.3*pow(1-t,3)+184.1*3*t*pow(1-t,2)+171.3*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
    }
    // outside curve
    iterationsAlongBezier =
(int)((313.9*randTile.spriteScale)/SimData.scale)/spacing);
    for(int i = 0; i < iterationsAlongBezier; i++) {
        double t =
((float)(i)/(float)(iterationsAlongBezier-1));
        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(218.6*pow(1-t,3)+211.2*3*t*pow(1-
t,2)+97.1*3*(1-t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
        (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
        (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,
        (float)(randTile.x_pos+(218.6*pow(1-t,3)+211.2*3*t*pow(1-t,2)+97.1*3*(1-
t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
        (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,
        (float)(randTile.x_pos+(218.6*pow(1-t,3)+211.2*3*t*pow(1-t,2)+97.1*3*(1-

```

```

t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(262.3*pow(1-t,3)+121.4*3*t*pow(1-t,2)+87.6*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
    }
    break;
    case 10:
        iterationsAlongBezier =
(int)((186.9*randTile.spriteScale)/SimData.scale)/spacing);
        // inside curve
        for(int i = 0; i < iterationsAlongBezier; i++) {
            double t =
((float)(i)/(float)(iterationsAlongBezier-1));
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(133.4*pow(1-t,3)+130.3*3*t*pow(1-
t,2)+56*3*(1-t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(0*pow(1-t,3)+78.2*3*t*pow(1-t,2)+90.8*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(133.4*pow(1-t,3)+130.3*3*t*pow(1-t,2)+56*3*(1-
t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(0*pow(1-t,3)+78.2*3*t*pow(1-t,2)+90.8*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,

```



```

t)*pow(t,2)+0*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(0*pow(1-t,3)+140.9*3*t*pow(1-t,2)+174.7*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
    }
    break;
    case 11:
        iterationsAlongBezier =
(int)((186.9*randTile.spriteScale)/SimData.scale)/spacing);
        // inside curve
        for(int i = 0; i < iterationsAlongBezier; i++) {
            double t =
((float)(i)/(float)(iterationsAlongBezier-1));
            randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(216.6*pow(1-t,3)+219.7*3*t*pow(1-
t,2)+294*3*(1-t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(0*pow(1-t,3)+78.2*3*t*pow(1-t,2)+90.8*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
  

        (float)(randTile.x_pos+(216.6*pow(1-t,3)+219.7*3*t*pow(1-t,2)+294*3*(1-
t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
  

        (float)(randTile.y_pos+(0*pow(1-t,3)+78.2*3*t*pow(1-t,2)+90.8*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,

```

```

        (float)(randTile.x_pos+(216.6*pow(1-t,3)+219.7*3*t*pow(1-t,2)+294*3*(1-
t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
    (float)(randTile.y_pos+(0*pow(1-t,3)+78.2*3*t*pow(1-t,2)+90.8*3*(1-
t)*pow(t,2)+90.1*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
    }
    // outside curve
    iterationsAlongBezier =
(int)((313.9*randTile.spriteScale)/SimData.scale)/spacing);
    for(int i = 0; i < iterationsAlongBezier; i++) {
        double t =
((float)(i)/(float)(iterationsAlongBezier-1));
        randTile.RoadCones.add( new RoadCone(
Geometry.createRectangle( (float)(randTile.x_pos+(131.4*pow(1-t,3)+138.8*3*t*pow(1-
t,2)+252.9*3*(1-t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,
        (float)(randTile.y_pos+(0*pow(1-t,3)+140.9*3*t*pow(1-t,2)+174.7*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale,
        (float)(randTile.x_pos+(131.4*pow(1-t,3)+138.8*3*t*pow(1-t,2)+252.9*3*(1-
t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
        (float)(randTile.y_pos+(0*pow(1-t,3)+140.9*3*t*pow(1-t,2)+174.7*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-
screen.getHeight()/2+size/2)/SimData.scale,

```

```

        (float)(randTile.x_pos+(131.4*pow(1-t,3)+138.8*3*t*pow(1-t,2)+252.9*3*(1-
t)*pow(t,2)+350*pow(t,3))*randTile.spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,
    }

    (float)(randTile.y_pos+(0*pow(1-t,3)+140.9*3*t*pow(1-t,2)+174.7*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*randTile.spriteScale-screen.getHeight()/2-
size/2)/SimData.scale ) );
}
break;
}
addTileFail = false;
}
else {
// Add Road Tile Failure
addTileFail = true;
}
} while(addTileFail == true);
return true;
}
/** This function sets the current map index of the user vehicle
 * to the corresponding parameters
 */
private void getMapTileIndex(float[] carCoords) {
    curMapX = (int)( carCoords[0] / (tileWidth*roadScale*SimData.scale) );
    curMapY = (int)( carCoords[1] / (tileHeight*roadScale*SimData.scale) );
}

public synchronized void update(float[] carCoords, float offX, float offY) {
    if((offsetX!=offX)|(offsetY!=offY))
        updateOffsets(offX, offY);
}

public void setScale(float newScale, float[] carCoords, float oldScale) {
    for(int i = 0; i < Map.size(); i++) {
        RoadTile tempTiletoScale = Map.get(i);
        tempTiletoScale.setScale(roadScale*SimData.scale, oldScale);
    }
}

public ArrayList<Line2D.Float> getAllRoadCenterLines() {
    ArrayList<Line2D.Float> allCenterLines = new
ArrayList<Line2D.Float>();
    for(int i=0; i < Map.size(); i++) {

```

```

        RoadTile tempTile = Map.get(i);
        allCenterLines.addAll(tempTile.getRoadCenterLines());
    }
    return allCenterLines;
}

public void alignVehicle(Car carToalign) {
    ArrayList<Line2D.Float> centerLines = new ArrayList<Line2D.Float>();
    float[] returnPt = new float[2];
    float smallestDistance = 0.0f;
    centerLines = getAllRoadCenterLines();

    for(int i=0; i<centerLines.size(); i++) {
        Line2D.Float centerLine = centerLines.get(i);
        float[] lineForm = Geometry.getLineFromPts( new float[] { (float)
        (-centerLine.getPt1().getX()-(screen.getWidth()/2)/SimData.scale)),
            (float)(-centerLine.getPt1().getY() -
        (screen.getHeight()/2)/SimData.scale) }, new float[] { (float)(-
        centerLine.getPt2().getX()-(screen.getWidth()/2)/SimData.scale)),
            (float)(-centerLine.getPt2().getY() -
        (screen.getHeight()/2)/SimData.scale) } );
        float[] alignedPt = Geometry.getClosestPttoPtandLine(lineForm,
        new float[] { (float)carToalign.getX(),(float)carToalign.getY() });
        double tempLength = sqrt( pow( carToalign.getX() - alignedPt[0],
        2) + pow(carToalign.getY() - alignedPt[1], 2) );
        if((tempLength < smallestDistance)||smallestDistance == 0.0f) {
            smallestDistance = (float)tempLength;
            returnPt = alignedPt;
        }
    }
    carToalign.setX(returnPt[0]);
    carToalign.setY(returnPt[1]);
}

public boolean generateManuever(Car vehicle) {
    String tempManuever = vehicle.getManuever().toLowerCase().trim();
    if(!tempManuever.equals("none")) {
        System.out.println("Building Path...");
        ArrayList<Line2D.Float> centerLines = new
ArrayList<Line2D.Float>();
        float[] nextPt = new float[2];
        float smallestDistance = 0f;
        centerLines = getAllRoadCenterLines();
        int centerLineindex = -1;

        for(int i=0; i<centerLines.size(); i++) {

```

```

        Line2D.Float centerLine = centerLines.get(i);
        float[] lineForm = Geometry.getLineFromPts( new float[] {
(float) -(centerLine.getP1().getX())+(screen.getWidth()/2)/SimData.scale),
                (float)(-(centerLine.getP1().getY() -
(screen.getHeight()/2)/SimData.scale)) }, new float[] { (float)(-
(centerLine.getP2().getX())+(screen.getWidth()/2)/SimData.scale),
                (float)(-(centerLine.getP2().getY() -
(screen.getHeight()/2)/SimData.scale)) } );

        float[] closetPt =
Geometry.getClosestPtToPtAndLine(lineForm,new float[] {
(float)vehicle.getX(),(float)vehicle.getY() });
        float tempLength = (float)sqrt( pow( vehicle.getX() -
closetPt[0], 2) + pow(vehicle.getY() - closetPt[1], 2) );
        if((tempLength<smallestDistance)||((i==0)) {
            smallestDistance = tempLength;
            nextPt = closetPt;
            centerLineindex = i;
        }
    }

    if(centerLineindex != -1) {
        Line2D.Float centerLine =
centerLines.get(centerLineindex);
        ArrayList<Line2D.Float> pathLines = new
ArrayList<Line2D.Float>();
        // For horizontal road sections
        if(tempManeuver.equals("slc")) {
            pathLines.add(new Line2D.Float(0f,(float)-
(centerLine.getP1().getY() - (screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale),
(float)(screen.getWidth()/2)/SimData.scale, (float)-(centerLine.getP1().getY() -
(screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale)));
            pathLines.add(new Line2D.Float((float)-
(screen.getWidth()/2)/SimData.scale,(float)-(centerLine.getP1().getY() -
(screen.getHeight()/2 - getRoadWidth()/4)/SimData.scale),0f ,(float)-
(centerLine.getP1().getY() - (screen.getHeight()/2 - getRoadWidth()/4)/SimData.scale)
));
        } else if(tempManeuver.equals("dlc")) {
            pathLines.add(new
Line2D.Float((float)(0.333f*(screen.getWidth()/SimData.scale)/2),(float)-
(centerLine.getP1().getY() - (screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale),
(float)(screen.getWidth()/2)/SimData.scale, (float)-(centerLine.getP1().getY() -
(screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale)));
            pathLines.add(new Line2D.Float((float)-
(screen.getWidth()/2)/SimData.scale,(float)-(centerLine.getP1().getY() -
(screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale),(float)-

```

```

        0.333f*(screen.getWidth()/SimData.scale)/2),(float)-(centerLine.getP1().getY() -
        (screen.getHeight()/2 + getRoadWidth()/4)/SimData.scale)));
                pathLines.add(new Line2D.Float((float)(-
        0.333f*(screen.getWidth()/SimData.scale)/2),(float)-(centerLine.getP1().getY() -
        (screen.getHeight()/2 -
        getRoadWidth()/4)/SimData.scale),(float)(0.333f*(screen.getWidth()/SimData.scale)/2)
        ,(float)-(centerLine.getP1().getY() - (screen.getHeight()/2 -
        getRoadWidth()/4)/SimData.scale)));
            } else if(tempManuever.equals("center")) {
                pathLines.add(new Line2D.Float((float)(-
        screen.getWidth()/2)/SimData.scale,(float)-(centerLine.getP1().getY() -
        (screen.getHeight()/2)/SimData.scale),
                (float)(screen.getWidth()/2)/SimData.scale,(float)-
        (centerLine.getP2().getY() - (screen.getHeight()/2)/SimData.scale) ));
            } else if(tempManuever.equals("sine")) {
                int numOfIters =
                (int)((screen.getWidth()/SimData.scale)/3.0f);
                for(int i = 0; i < numOfIters-1; i++) {
                    pathLines.add(new Line2D.Float((float)((-
        screen.getWidth()/2)/SimData.scale-i*3.0f), (float)-
        (centerLine.getP1().getY())+5*sin(10*i) - (screen.getHeight()/2)/SimData.scale),(float)((-
        screen.getWidth()/2)/SimData.scale-(i+1)*3.0f),(float)-
        (centerLine.getP1().getY())+5*sin(10*(i+1)) - (screen.getHeight()/2)/SimData.scale)));
                }
            }
            vehicle.setPath(pathLines);
        }
    }
    return true;
}

public ArrayList<float[]> getAllRoadCones() {
    ArrayList<float[]> returnObstacles = new ArrayList<float[]>();
    for(int i = 0; i < Map.size(); i++) {
        RoadTile tempTile = Map.get(i);
        returnObstacles.addAll( tempTile.getAllRoadCones() );
    }
    return returnObstacles;
}

public ArrayList<float[]> getRoadConesInRange(float sensor_range, float[] carPos) {
    ArrayList<float[]> returnObstacles = new ArrayList<float[]>();
    for(int i = 0; i < Map.size(); i++) {
        RoadTile tempTile = Map.get(i);

```

```

        returnObstacles.addAll(
tempTile.getRoadConesInRange(sensor_range,carPos) );
    }
    return returnObstacles;
}

private class RoadTile extends Sprite {
    protected int          tileID;
    protected float         x_pos;
    protected float         y_pos;
    protected int          mapX;
    protected int          mapY;
    protected boolean      rotate180;
    protected ArrayList<Line2D.Float> centerLines = new
ArrayList<Line2D.Float>();
    protected boolean[]     RoadRules; // [ rotateAllowed, top connection, bottom
connection, left connection, right connection ]
    protected ArrayList<RoadCone> RoadCones = new
ArrayList<RoadCone>();

    public RoadTile(Dimension screen, float scaler, String filePath, int mapX,
int mapY, boolean rotate180) {
        super(screen,roadScale*SimData.scale,filePath,false);

        this.mapX = mapX;
        x_pos = mapX*spriteWidth*spriteScale;
        this.mapY = mapY;
        y_pos = mapY*spriteHeight*spriteScale;
        this.rotate180 = rotate180;
        // up, down, left, right
        switch(filePath) {
            case fourWayIntersection:
                tileID = 0;
                RoadRules = new boolean[] { false, true, true, true,
true };
                centerLines.add(new Line2D.Float(
x_pos/SimData.scale,
(mapY*spriteHeight*spriteScale+(spriteHeight*spriteScale)/2)/SimData.scale,(x_pos+sp
riteWidth*spriteScale)/SimData.scale,(mapY*spriteHeight*spriteScale+(spriteHeight*spr
iteScale)/2)/SimData.scale ) );
                centerLines.add(new Line2D.Float(
(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,y_pos/SimDa
ta.scale,(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,(y_po
s+spriteHeight*spriteWidth)/SimData.scale ) );
                break;
            case fourWay_HorizRitAway:

```

```

        tileID = 3;
        RoadRules = new boolean[] { false, true, true, true,
true };
        centerLines.add(new Line2D.Float(
x_pos/SimData.scale,
(mapY*spriteHeight*spriteScale+(spriteHeight*spriteScale)/2)/SimData.scale,(x_pos+sp
riteWidth*spriteScale)/SimData.scale,(mapY*spriteHeight*spriteScale+(spriteHeight*spr
iteScale)/2)/SimData.scale ) );
        centerLines.add(new Line2D.Float(
(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,y_pos/SimDa
ta.scale,(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,(y_po
s+spriteHeight*spriteWidth)/SimData.scale ) );
        break;
    case fourWay_VertRitAway:
        tileID = 4;
        RoadRules = new boolean[] { false, true, true, true,
true };
        centerLines.add(new Line2D.Float(
x_pos/SimData.scale,
(mapY*spriteHeight*spriteScale+(spriteHeight*spriteScale)/2)/SimData.scale,(x_pos+sp
riteWidth*spriteScale)/SimData.scale,(mapY*spriteHeight*spriteScale+(spriteHeight*spr
iteScale)/2)/SimData.scale ) );
        centerLines.add(new Line2D.Float(
(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,y_pos/SimDa
ta.scale,(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,(y_po
s+spriteHeight*spriteWidth)/SimData.scale ) );
        break;
    case Tee_HorizRitAway:
        tileID = 6;
        RoadRules = new boolean[] { true, false, true, true,
true };
        centerLines.add(new Line2D.Float(
x_pos/SimData.scale,
(mapY*spriteHeight*spriteScale+(spriteHeight*spriteScale)/2)/SimData.scale,(x_pos+sp
riteWidth*spriteScale)/SimData.scale,(mapY*spriteHeight*spriteScale+(spriteHeight*spr
iteScale)/2)/SimData.scale ) );
        break;
    case Tee_VertRitAway:
        tileID = 7;
        RoadRules = new boolean[] { true, true, true, true,
false };
        centerLines.add(new Line2D.Float(
(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,y_pos/SimDa
ta.scale,(mapX*spriteWidth*spriteScale+spriteWidth*spriteScale/2)/SimData.scale,(y_po
s+spriteHeight*spriteWidth)/SimData.scale ) );
        break;

```

```

        case Tee_HorizUpRitAway:
            tileID = 12;
            RoadRules = new boolean[] { false, true, false, true,
true };
                break;
        case Tee_VertRightRitAway:
            tileID = 13;
            RoadRules = new boolean[] { false, true, true, false,
true };
                break;
        case LeftCurveRoad:
            tileID = 8;
            RoadRules = new boolean[] { true, false, true, false,
true };
                // curve
                int iterationsAlongBezier =
(int)((((186.9*spriteScale)/SimData.scale)/spacing);
                    for(int i = 0; i < iterationsAlongBezier-1; i++) {
                        double t =
((float)(i)/(float)(iterationsAlongBezier-1));
                        double tplus =
((float)(i+1)/(float)(iterationsAlongBezier-1));
                        centerLines.add(new Line2D.Float(
(float)(x_pos+(216.6*pow(1-t,3)+219.7*3*t*pow(1-t,2)+294.3*3*(1-
t)*pow(t,2)+350*pow(t,3))*spriteScale)/SimData.scale,
                            (float)(y_pos+(262.3*pow(1-t,3)+184.1*3*t*pow(1-t,2)+171.3*3*(1-
t)*pow(t,2)+172.2*pow(t,3))*spriteScale)/SimData.scale,
                                (float)(x_pos+(216.6*pow(1-tplus,3)+219.7*3*tplus*pow(1-
tplus,2)+294.3*3*(1-tplus)*pow(tplus,2)+350*pow(tplus,3))*spriteScale)/SimData.scale,
                                    (float)(y_pos+(262.3*pow(1-tplus,3)+184.1*3*tplus*pow(1-
tplus,2)+171.3*3*(1-
tplus)*pow(tplus,2)+172.2*pow(tplus,3))*spriteScale)/SimData.scale));
                        }
                        break;
        case RightCurveRoad:
            tileID = 9;
            RoadRules = new boolean[] { true, false, true, true,
false };
                break;
        case LeftUpCurveRoad:
            tileID = 10;
            RoadRules = new boolean[] { true, true, false, true,
false };
                break;
    }
}

```

```

        break;
    case RightUpCurveRoad:
        tileID = 11;
        RoadRules = new boolean[] { true, true, false, false,
true };
        break;
    case HorizStraightRoad:
        tileID = 1;
        RoadRules = new boolean[] { false, false, false,
true, true };
        centerLines.add(new Line2D.Float(
x_pos/SimData.scale,
(mapY*spriteHeight*spriteScale+(spriteHeight*spriteScale)/2)/SimData.scale,(x_pos+sp
riteWidth*spriteScale)/SimData.scale,(mapY*spriteHeight*spriteScale+(spriteHeight*spr
iteScale)/2)/SimData.scale ) );
        break;
    case VertStraightRoad:
        tileID = 2;
        RoadRules = new boolean[] { false, true, true, false,
false };
        centerLines.add(new Line2D.Float(
(mapX*spriteWidth*spriteScale+(spriteWidth*spriteScale)/2+0.75f*getRoadHeight())/Si
mData.scale,y_pos/SimData.scale,(mapX*spriteWidth*spriteScale+(spriteWidth*spriteS
cale)/2+0.75f*getRoadHeight())/SimData.scale,(y_pos+spriteHeight*spriteScale)/SimDat
a.scale ) );
        break;
    case GrassRoad:
        tileID = 5;
        RoadRules = new boolean[] { false, false, false,
false, false };
        break;
    }
}

public ArrayList<Line2D.Float> getRoadCenterLines() {
    return centerLines;
}

public void draw(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    AffineTransform affineTransform = new AffineTransform();
    affineTransform.translate( x_pos+offsetX, y_pos+offsetY );
    affineTransform.scale( spriteScale, spriteScale );
    if(rotate180 == true)
        affineTransform.rotate(PI, spriteWidth/2, spriteHeight/2);
    g2d.drawImage( getImage(), affineTransform, null );
}

```

```

        for(int i=0; i < RoadCones.size(); i++) {
            RoadCone tempCone = RoadCones.get(i);
            tempCone.draw(g2d);
        }

        g2d.translate(offsetX, offsetY);
        g2d.scale(SimData.scale,SimData.scale);
        g2d.scale(1/SimData.scale,1/SimData.scale);
        g2d.translate(-offsetX, -offsetY);
    }

    public ArrayList<float[]> getAllRoadCones() {
        ArrayList<float[]> tempConeEntities = new ArrayList<float[]>();
        for(int i=0; i < RoadCones.size(); i++) {

            tempConeEntities.add(RoadCones.get(i).getBoundaryPoints());
        }
        return tempConeEntities;
    }

    public ArrayList<float[]> getRoadConesInRange(float sensor_range,
float[] carPos) {
        ArrayList<float[]> tempConeEntities = new ArrayList<float[]>();

        for(int i=0; i < RoadCones.size(); i++) {
            RoadCone tempCone = RoadCones.get(i);
            float[] conePostemp = tempCone.getConePosition();
            if(Geometry.length(conePostemp[0],conePostemp[1],
carPos[0], carPos[1]) <= sensor_range)

            tempConeEntities.add(tempCone.getBoundaryPoints());
        }

        return tempConeEntities;
    }

    public void setMapX(int x) {
        mapX = x;
        x_pos = mapX*spriteWidth*spriteScale;
    }

    public void setMapY(int y) {
        mapY = y;
        y_pos = mapY*spriteHeight*spriteScale;
    }
}

```

```

public void setScale(float newScale, float oldScale) {

    spriteScale = newScale;

    float new_x_pos = mapX*spriteWidth*spriteScale;
    float new_y_pos = mapY*spriteHeight*spriteScale;

    x_pos = new_x_pos;
    y_pos = new_y_pos;

    // Shift road cones
    int coneCounter;
    switch(tileID) {
        case 1:
            coneCounter = 0;
            for(int i = 0; i < RoadCones.size(); i+=2) {
                RoadCone tempCone = RoadCones.get(i);
                float[] tempBoundaryPts =
tempCone.getBoundaryPoints();

                tempCone.setBoundaryPoints(
Geometry.createRectangle( (float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2-size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*spriteScale-size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-screen.getWidth()/2-
size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-screen.getHeight()/2-
(getRoadWidth()/2)*spriteScale+size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*spriteScale+size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2-(getRoadWidth()/2)*spriteScale-size/2)/SimData.scale );

                coneCounter++;
            }
            coneCounter = 0 ;
            for(int i = 1; i < RoadCones.size(); i+=2) {
                RoadCone tempCone = RoadCones.get(i);
                float[] tempBoundaryPts =
tempCone.getBoundaryPoints();

```

```

tempCone.setBoundaryPoints(
Geometry.createRectangle((float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2-size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*spriteScale-size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-screen.getWidth()/2-
size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*spriteScale+size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*spriteScale+size/2)/SimData.scale,
(float)(x_pos+spacing*SimData.scale*coneCounter-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos+(tileHeight/2)*spriteScale-
screen.getHeight()/2+(getRoadWidth()/2)*spriteScale-size/2)/SimData.scale ) );

coneCounter++;
}
break;
case 2:
coneCounter = 0;
for(int i = 0; i < RoadCones.size(); i+=2) {
RoadCone tempCone = RoadCones.get(i);
float[] tempBoundaryPts =
tempCone.getBoundaryPoints();

tempCone.setBoundaryPoints(
Geometry.createRectangle( (float)(x_pos+(tileWidth/2)*spriteScale-
(getRoadHeight()/2)*spriteScale-screen.getWidth()/2-
size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter-size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale-(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2-size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter+size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale-(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter+size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale-(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter-size/2)/SimData.scale ) );

coneCounter++;
}

```

```

        }
        coneCounter = 0 ;
        for(int i = 1; i < RoadCones.size(); i+=2) {
            RoadCone tempCone = RoadCones.get(i);
            float[] tempBoundaryPts =
tempCone.getBoundaryPoints();

            tempCone.setBoundaryPoints(
Geometry.createRectangle((float)(x_pos+(tileWidth/2)*spriteScale+(getRoadHeight()/2)
*spriteScale-screen.getWidth()/2-size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter-size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale+(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2-size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter+size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale+(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter+size/2)/SimData.scale,
(float)(x_pos+(tileWidth/2)*spriteScale+(getRoadHeight()/2)*spriteScale-
screen.getWidth()/2+size/2)/SimData.scale,(float)(y_pos-
screen.getHeight()/2+spacing*SimData.scale*coneCounter-size/2)/SimData.scale ) );

            coneCounter++;
        }
        break;
    }
    refreshMap();
}

public void setRotate(boolean rotate) {
    rotate180 = true;
}
}

private class RoadCone {
    protected float[] conePos = new float[2];
    protected float[] boundaryPoints;

    public RoadCone(float[] boundaryPoints) {
        this.boundaryPoints = boundaryPoints;
        conePos =
Geometry.computePolygonCentroid(boundaryPoints);
    }
}

```

```

        public void draw(Graphics2D g2d) {
            // Translate to global frame origin (center of screen)
            g2d.translate(screen.getWidth()/2+ offsetX,
screen.getHeight()/2+ offsetY);
            // Scale viewport according to current simulation scale
            g2d.scale(SimData.scale, SimData.scale);
            drawGeneralPath( g2d, boundaryPoints );
            g2d.scale((1/SimData.scale), (1/SimData.scale));
            g2d.translate(-screen.getWidth()/2-offsetX, -
screen.getHeight()/2-offsetY);
        }

        public void setBoundaryPoints(float[] boundaryPoints) {
            this.boundaryPoints = boundaryPoints;
            conePos =
Geometry.computePolygonCentroid(boundaryPoints);
        }

        protected float[] getBoundaryPoints() {
            return boundaryPoints;
        }

        protected float[] getConePosition() {
            return conePos;
        }
    }
}

```

vdhils/SerialCommunicationJSSC.java:

```

/**
 *      \file SerialCommunicationJSSC.java
 *      \brief SerialCommunication is a class that performs serial tasks in order to
communicate with the BeagleBone embedded device by managing setup, listening, and
writing data to the serial port.
*
*      References:
*
*      Revisions:
*          \li 4/18/2014 TFS
*
*      License:
*      This file is copyright 2014 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/

```

```

/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package vdhils;
// Import java API's/libraries
import java.io.InputStream;
import java.io.*;
import jssc.SerialPort;
import jssc.SerialPortList;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.*;
import java.nio.ByteBuffer;
/**  \class SerialCommunication
 *   \brief SerialCommunication is a class that performs serial tasks in order to
communicate with the BeagleBone embedded device by managing setup, listening, and
writing data to the serial port.
*/
public class SerialCommunicationJSSC implements Runnable, Serializable,
SerialPortEventListener {
    /** Serializable UID for saving/loading functionality */
```

```

private static final long serialVersionUID = 7989078461104748972L;
/** Serial port object */
private SerialPort serialPort;
/** Buffers for x and y point-cloud data sent to embedded hardware for
processing */
    private volatile byte[] bufferX, bufferY; // outgoing
data - accessed by multiple threads need to be volatile
    /** Buffer for steerangle sent to embedded hardware for processing */
    private volatile byte steerAngle;
    // outgoing data - accessed by multiple threads need to be volatile
    /** Buffer for throttle sent to embedded hardware for processing */
    private volatile byte throttle; ///
outgoing data - accessed by multiple threads need to be volatile
    /** Vehicle x-y position & angle (yaw) buffer sent to embedded hardware for
processing */
    private volatile byte[] bufferPos_x, bufferPos_y, bufferAng;// outgoing data -
accessed by multiple threads need to be volatile
    /** Vehicle speed from simulation */
    private volatile byte[] speed_sim;
    /** Measured speed from embedded hardware utilizing wheel encoders (average
of 4 wheel encoders is returned) */
    protected volatile float speed;
    // incoming data - accessed by multiple threads need to be volatile
    /** RRT enabled flag to signal use of RRT generated steer angles */
    protected volatile int RRT_ENABLE;
    // incoming data - accessed by multiple threads need to be volatile
    /** Calculated RRT steer angle to avoid collision and follow safe path */
    protected volatile float rrtSteer;
    // incoming data (time, steer angle) - accessed by multiple threads need to be
volatile
        /// \todo Allow for RRT steer data as table
        /// \todo If embedded hardware if not available don't send or start this runnable in
new thread - instead start client to talk to server running embedded software
    private String PORT_NAMES[] = {
        ""
    };
    private String PORT_NAME = "";
    private InputStream input;
    private OutputStream output;
    private int TIME_OUT = 3000;
    private int DATA_RATE = 3000000;

    public volatile boolean processingData; // accessed by multiple threads need to be
volatile
    // Some useful ASCII values

```

```

public final static byte SPACE_ASCII = 32;
public final static byte DASH_ASCII = 45;
public final static byte NEW_LINE_ASCII = 10;

public SerialCommunicationJSSC(String searchPorts[],int dataRate,int timeOut) {
    this(searchPorts, dataRate);
    TIME_OUT = timeOut;
}
public SerialCommunicationJSSC(String searchPorts[],int dataRate) {
    this(searchPorts);
    DATA_RATE = dataRate;
}
public SerialCommunicationJSSC(String searchPorts[]) {
    PORT_NAMES = searchPorts;
}
public SerialCommunicationJSSC(String comPort) {
    PORT_NAME = comPort;
}
public SerialCommunicationJSSC() {

}
public float getMeasuredSpeed() {
    return speed;
}
public int getRRT_Enable() {
    return RRT_ENABLE;
}
public float getRRT_Steer() {
    return rrtSteer;
}
private void addSerialPortEventListener() {
    try {
        int mask = SerialPort.MASK_RXCHAR;// +
SerialPort.MASK_CTS + SerialPort.MASK_DSR;
        serialPort.setEventsMask(mask);
        serialPort.addEventListerner(this);
    } catch(SerialPortException ex) {
        System.err.println(ex.toString());
        SimData.logger.warning("Serial connection to embedded hardware
failed: "+ex.toString());
    }
}
public void setBufferXFloat(ArrayList<Float> newXBuffer) {
    short[] tempconv = new short[newXBuffer.size()];
    // System.out.println("X Size: "+newXBuffer.size());
    bufferX = new byte[newXBuffer.size()*2+1];
}

```

```

        for(int i = 0; i < newXBuffer.size(); i++) {
            tempconv[i] = (short)Math.round(newXBuffer.get(i)*100);
            bufferX[i*2] = (byte)(tempconv[i]);
            bufferX[i*2+1] = (byte)(tempconv[i] >> 8);
            // System.out.println(newXBuffer.get(i));
        }
    }
    public void setBufferYFloat(ArrayList<Float> newYBuffer) {
        // this.bufferX = newXBuffer;
        // Initialize new short array to resolve float to a short to send over serial
        with 2 decimal place precision
        short[] tempconv = new short[newYBuffer.size()];
        // System.out.println("Y Size: "+newYBuffer.size());
        // Initialize buffer to new byte array with appropriate size after converting
        each float to a byte
        bufferY = new byte[newYBuffer.size()*2+1];
        for(int i = 0; i < newYBuffer.size(); i++) {
            // Scale float to an integer by multiplying by 100
            tempconv[i] = (short)Math.round(newYBuffer.get(i)*100);
            bufferY[i*2] = (byte)(tempconv[i]);
            bufferY[i*2+1] = (byte)(tempconv[i] >> 8);
            // System.out.println(newYBuffer.get(i));
        }
    }
    public void setBufferXDouble(ArrayList<Double> newXBuffer) {
        short[] tempconv = new short[newXBuffer.size()];
        // System.out.println("Size: "+newXBuffer.size());
        bufferX = new byte[newXBuffer.size()*2+1];
        for(int i = 0; i < newXBuffer.size(); i++) {
            tempconv[i] = (short)Math.round(-newXBuffer.get(i)*328);
            bufferX[i*2] = (byte)(tempconv[i]);
            bufferX[i*2+1] = (byte)(tempconv[i] >> 8);
            //System.out.println(newXBuffer.get(i));
        }
    }
    public void setBufferYDouble(ArrayList<Double> newYBuffer) {
        short[] tempconv = new short[newYBuffer.size()];
        // System.out.println("Size: "+newYBuffer.size());
        bufferY = new byte[newYBuffer.size()*2+1];
        for(int i = 0; i < newYBuffer.size(); i++) {
            tempconv[i] = (short)Math.round(newYBuffer.get(i)*328);
            bufferY[i*2] = (byte)(tempconv[i]);
            bufferY[i*2+1] = (byte)(tempconv[i] >> 8);
            //System.out.println(newYBuffer.get(i));
        }
    }
}

```

```

public void setSteerAngle(byte steerAngle) {
    this.steerAngle = steerAngle;
}
public void setThrottle(byte throttle) {
    this.throttle = throttle;
}
public void setSimSpeed(float SimSpeed) {
    short tempconv = (short) Math.round(SimSpeed);
    speed_sim = new byte[2];

    speed_sim[0] = (byte) tempconv;
    speed_sim[1] = (byte) (tempconv >> 8);
}
public void setBufferPosX(float pos_X) {
    short tempconv = (short) Math.round(pos_X*100);
    bufferPos_x = new byte[2];
    // Little Endian?
    bufferPos_x[0] = (byte) tempconv;
    bufferPos_x[1] = (byte) (tempconv >> 8);
}
public void setBufferPosY(float pos_Y) {
    short tempconv = (short) Math.round(pos_Y*100);
    bufferPos_y = new byte[2];
    // Little Endian?
    bufferPos_y[0] = (byte) tempconv;
    bufferPos_y[1] = (byte) (tempconv >> 8);
}
public void setBufferAng(float angle) {
    short tempconv = (short) Math.round(angle*100);
    bufferAng = new byte[2];
    // Little Endian?
    bufferAng[0] = (byte) tempconv;
    bufferAng[1] = (byte) (tempconv >> 8);
}
public static byte[] toByteArray(double value) {
    byte[] bytes = new byte[8];
    ByteBuffer.wrap(bytes).putDouble(value);
    return bytes;
}
public static byte[] toByteArray(float value) {
    byte[] bytes = new byte[4];
    ByteBuffer.wrap(bytes).putFloat(value);
    return bytes;
}
public void connect() {
    System.out.println("Connecting...");
}

```

```

        SimData.logger.info("Connecting to embedded hardware serial port at:
"+this.getClass().getName());

        if((PORT_NAME != null) && (PORT_NAME != ""))
            serialPort = new SerialPort(PORT_NAME);
        else
            serialPort = new SerialPort(PORT_NAMES[0]);

        try{
            serialPort.openPort();//Open serial port
            serialPort.setParams(DATA_RATE,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);//Set params. Also you can set params
by this string: serialPort.setParams(9600, 8, 1, 0);

            System.out.println("Connected ");
            SimData.logger.info("Connected and awaiting serial input on port:
"+this.getClass().getName());
            } catch (Exception e) {
                System.err.println(e.toString());
                SimData.logger.warning("Serial connection to embedded hardware
failed: "+e.toString());
            }
            addSerialPortEventListener();
        }
    public void disconnect() {
        System.out.println("Disconnecting...");

        try {
            serialPort.closePort();//Close serial port
            System.out.println("System Status: Closing serial connection");
        }
        catch (Exception exe) {
            SimData.logger.severe("Disconnect from Embedded Hardware Failed!
("+exe.toString()+")");
        }
    }
    public void run() {
        try {
            long startTime = System.nanoTime();
            // Write a byte to tell the serialPort how many bytes to read
            // short numBytes = (short)(10+bufferX.length+bufferY.length);
            short numBytes = (short)(8+bufferX.length+bufferY.length);
            byte[] outputnumBytes = new byte[2];
            outputnumBytes[0] = (byte)numBytes;

```

```

        outputnumBytes[1] = (byte)(numBytes >> 8);
        serialPort.writeBytes(outputnumBytes);
        // byte[] outputBytes = new
byte[10+bufferX.length+bufferY.length];
        byte[] outputBytes = new byte[8+bufferX.length+bufferY.length];
        outputBytes[0] = steerAngle;
        // System.out.println("Steer Out: "+steerAngle);
        outputBytes[1] = throttle;
        // System.out.println("Throttle Out: "+throttle);

        System.arraycopy(bufferPos_x,0,outputBytes,2,bufferPos_x.length);

        System.arraycopy(bufferPos_y,0,outputBytes,4,bufferPos_y.length);
        System.arraycopy(bufferAng,0,outputBytes,6,bufferAng.length);
        System.arraycopy(bufferX,0,outputBytes,8,bufferX.length);

        System.arraycopy(bufferY,0,outputBytes,8+bufferX.length,bufferY.length);
        // System.arraycopy(speed_sim,0,outputBytes,8,speed_sim.length);
        // System.arraycopy(bufferX,0,outputBytes,10,bufferX.length);
        //

        System.arraycopy(bufferY,0,outputBytes,10+bufferX.length,bufferY.length);

        // write output array to serial port
        serialPort.writeBytes(outputBytes);
        // System.out.println();
        // System.out.println("Serial Data Send Transfer Elasped Time:
"+(System.nanoTime()-startTime)/1000000000.0);
        // System.out.println();

    } catch(Exception e) { }
    //processingData = false;
}

public static String[] getSerialPorts() {
    return SerialPortList.getPortNames();
}

public void serialEvent(SerialPortEvent event) {
    // System.out.println("Serial data event");
    try {
        // OLD - Tested
        byte buffer[] = serialPort.readBytes(5);
        // for(int i = 0; i < buffer.length; i++) {
            // System.out.println("Serial data received: "+buffer[i]);
        // }
        // Two first bytes is the measured speed from the embedded
hardware times a factor of 100 (limit to 2 decimal places for speed!)
        int result = buffer[0]&0xff | (buffer[1] << 8);
    }
}

```

```

speed = result/100.0f;
speed = 0.44704f*speed;
// System.out.println("Speed: "+speed);

// next byte is rrt enable flag and number of elements in steer time
lookup table ( MSB is rrt enable flag and all other bits are # of elements in table )
RRT_ENABLE = buffer[2]; // Wasting bits of data
// the remaining bytes are the rrt steer and throttle commands as a
function of time (quantized lookup table) and the number of values in the lookup table
(the number to read...)
// for now just get the first steer command (same as system
concluding Technical Conference in Korea) but update to use
result = buffer[3]&0xff | (buffer[4] << 8);
rrtSteer = -result/100.0f;

// NEW - Untested
// read first 2 bytes -> number of subsequent bytes to read
// byte bufferRead[] = serialPort.readBytes(2);
// int numBytes = bufferRead[0]&0xff | (bufferRead[1] << 8);
// if(numBytes <= 5) {
    // byte buffer[] = serialPort.readBytes(5);
    // Two first bytes is the measured speed from the embedded
hardware times a factor of 100 (limit to 2 decimal places for speed!)
    // int result = buffer[0]&0xff | (buffer[1] << 8);
    // speed = result/100.0f;
    // speed = 0.44704f*speed;
    // next byte is rrt enable flag and number of elements in
steer time lookup table ( MSB is rrt enable flag and all other bits are # of elements in
table )
    // RRT_ENABLE = buffer[2]; // Wasting bits of data
    // the remaining bytes are the rrt steer and throttle
commands as a function of time (quantized lookup table) and the number of values in the
lookup table (the number to read...)
// for now just get the first steer command (same as system
concluding Technical Conference in Korea) but update to use
// result = buffer[3]&0xff | (buffer[4] << 8);
// rrtSteer = -result/100.0f;
// } else {
    // byte buffer[] = serialPort.readBytes(numBytes);
    // Two first bytes is the measured speed from the embedded
hardware times a factor of 100 (limit to 2 decimal places for speed!)
    // int result = buffer[0]&0xff | (buffer[1] << 8);
    // speed = result/100.0f;
    // speed = 0.44704f*speed;

```

```

        // next byte is rrt enable flag and number of elements in
        steer time lookup table ( MSB is rrt enable flag and all other bits are # of elements in
        table )
        // RRT_ENABLE = buffer[2]; // Wasting bits of data
        // the remaining bytes are the rrt steer and throttle
commands as a function of time (quantized lookup table) and the number of values in the
lookup table (the number to read...)
        // for now just get the first steer command (same as system
concluding Technical Conference in Korea) but update to use
        // result = buffer[3]&0xff | (buffer[4] << 8);
        // rrtSteer = -result/100.0f;
        // If more than 5 bytes read in data to allow drawing of
controller's output
        // Read in obstacle data

        // Read in road line data

        // Read in final path data
    // }

processingData = false;      // signals to simulation that send and
receive is complete - continue updating. (this event errors on the EDT)
} catch (SerialPortException ex) {
    SimData.logger.severe("Failed to read serial event from embedded
hardware");
}
}
}
}


```

vdhils/SettingsData.java:

```

/**
 *      \file SettingsData.java
 *      \brief Settings Data class
 *
 *      Revisions:
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE

```

\* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
PARTICULAR PURPOSE  
\* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  
CONTRIBUTORS BE  
\* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
EXEMPLARY, OR CONSEQUEN-  
\* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
SUBSTITUTE GOODS  
\* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
INTERRUPTION) HOWEVER  
\* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,  
STRICT LIABILITY,  
\* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY  
WAY OUT OF THE USE  
\* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
DAMAGE. \*/

\*\*\*\*\*

```
package vdhils;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
import java.lang.reflect.InvocationTargetException;
import java.util.Random;
import java.util.concurrent.*;
import java.io.*;
import java.util.Hashtable;
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
```

```

import at.wisch.joystick.ffeffect.direction.*;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.border.EtchedBorder;
/**  \class SettingsData
 *   \brief Settings Data class
 */
public class SettingsData implements Serializable {
    /** Serializable UID for saving/loadding functionality */
    private static final long serialVersionUID = 7989078461104748974L;
    private GraphicsSettings graphicsSettings;
    private ControllerSettings controllerSettings;
    // private SimulationSettings simulationSettings;
    private EmbeddedSettings embeddedSettings;

    public SettingsData() {
        graphicsSettings = new GraphicsSettings();
        controllerSettings = new ControllerSettings();
        // simulationSettings = new SimulationSettings();
        embeddedSettings = new EmbeddedSettings();
    }
    public GraphicsSettings getGraphicsSettings() {
        return graphicsSettings;
    }
    public ControllerSettings getJoystickSettings() {
        return controllerSettings;
    }
    // public SimulationSettings getSimulationSettings() {
    //     return simulationSettings;
    // }
    public EmbeddedSettings getEmbeddedSettings() {
        return embeddedSettings;
    }
    protected void setJoystick( FFJoystick joystick ) {

    }
    private class GraphicsSettings extends JPanel implements ItemListener,
    ActionListener {
        private Color longVelSignalColor;
        private Color latVelSignalColor;
        private Color speedSignalColor;
        private Color steerAngleSignalColor;
        private Color angVelSignalColor;
        private Color trailColor;
        private Color obstacleTrailColor;
    }
}

```

```

        private final Color[] colors = { Color.black, Color.blue,
Color.cyan, Color.darkGray,
                                Color.gray, Color.green, Color.lightGray,
Color.magenta,
                                Color.orange, Color.pink, Color.red, Color.white,
Color.yellow };

        private JCheckBox plotSpeedometer;
        private JCheckBox plotTrail;
        private JCheckBox plotRealTimeData;
        private final JComboBox longVelocityColor = new
JComboBox(colors);
        private final JComboBox latVelColor = new JComboBox(colors);

        private JCheckBox plotLidarScan;
        private JCheckBox plotSegmentedObstacles;
        private JCheckBox plotThreateningObstacles;
        private JCheckBox plotCarTexture;
        private JCheckBox plotCarWheelTexture;

        private boolean renderCarTextures;
        private boolean renderWheelTextures;
        private boolean renderSpeedometer;

        private boolean realTimePlotterEnabled;
        private boolean realTimeDataLegend;
        private boolean renderLidarScan;
        private boolean renderObstacles;
        private boolean renderThreateningObstacles;
        private boolean renderTrail;
        private boolean obstacleTrailEnabled;

        private double realTimePlotterScale;
        private double speedometerScale;
        private double trailSize;
        private double obstacleTrailSize;

        private int trailLength;
        private int obstacleTrailLength;
        private int carGraphic;

    public GraphicsSettings() {

        setBackground(SimData.CalPoly_CREAM);
        setLayout(new GridBagLayout());

```

```

        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.CENTER;

        renderSpeedometer = true;
        realTimePlotterEnabled = true;
        renderLidarScan = true;
        renderObstacles = false;
        renderThreateningObstacles = false;
        renderTrail = true;
        renderCarTextures = true;
        renderWheelTextures = true;

        longVelocityColor.setMaximumRowCount(5);
        // longVelocityColor.setEditable(true);
        longVelocityColor.setRenderer(new
ColorComboBoxEditor());
        longVelocityColor.setOpaque(true);
        // longVelocityColor.setLightWeightPopupEnabled(false);
        longVelocityColor.setFocusable(true);
        longVelocityColor.setVisible(true);
        longVelocityColor.addActionListener(this);

        plotSpeedometer = new JCheckBox("Plot Speedometer");

        //plotSpeedometer.setBackground(SimData.CalPoly_CREAM);
        plotSpeedometer.setSelected( renderSpeedometer );

        plotTrail = new JCheckBox("Plot Trail");
        //plotTrail.setBackground(SimData.CalPoly_CREAM);
        plotTrail.setSelected( renderTrail );

        plotRealTimeData = new JCheckBox("Plot Real Time
Data");
        //plotRealTimeData.setBackground(SimData.CalPoly_CREAM);
        plotRealTimeData.setSelected( realTimePlotterEnabled );

        plotLidarScan = new JCheckBox("Plot Lidar Scan Data");

        //plotLidarScan.setBackground(SimData.CalPoly_CREAM);
        plotLidarScan.setSelected( renderLidarScan );

        plotSegmentedObstacles = new JCheckBox("Plot
Segmented Obstacles from Lidar data");

        //plotSegmentedObstacles.setBackground(SimData.CalPoly_CREAM);

```

```

        plotSegmentedObstacles.setSelected( renderObstacles );

        plotSegmentedObstacles.setEnabled(false);

        plotThreateningObstacles = new JCheckBox("Plot
Threatening Obstacles from Lidar data");

//plotThreateningObstacles.setBackground(SimData.CalPoly_CREAM);
        plotThreateningObstacles.setSelected(
renderThreateningObstacles );

        plotThreateningObstacles.setEnabled(false);

        plotCarTexture = new JCheckBox("Plot Car Texture
(setting to false renders a wireframe model)");

//plotCarTexture.setBackground(SimData.CalPoly_CREAM);
        plotCarTexture.setSelected( renderCarTextures );

        plotCarWheelTexture = new JCheckBox("Plot Tire Texture
(setting to false renders a wireframe model)");

//plotCarWheelTexture.setBackground(SimData.CalPoly_CREAM);
        plotCarWheelTexture.setSelected( renderWheelTextures );

        plotSpeedometer.addItemListener( this );
        plotTrail.addItemListener( this );
        plotRealTimeData.addItemListener( this );
        plotLidarScan.addItemListener( this );
        plotSegmentedObstacles.addItemListener( this );
        plotThreateningObstacles.addItemListener( this );
        plotCarTexture.addItemListener( this );
        plotCarWheelTexture.addItemListener( this );

        /// \todo Set components not visible depending on selected
checked items

JPanel checkPanel = new JPanel( new GridLayout(0,1) );
//checkPanel.setBackground(SimData.CalPoly_CREAM);
checkPanel.add( new JLabel("Render Settings:") );
checkPanel.add( plotSpeedometer );
checkPanel.add( plotTrail );
checkPanel.add( plotRealTimeData );
checkPanel.add( longVelocityColor );
checkPanel.add( plotLidarScan );
checkPanel.add( plotSegmentedObstacles );

```

```

        checkPanel.add( plotThreateningObstacles );
        checkPanel.add( plotCarTexture );
        checkPanel.add( plotCarWheelTexture );
        checkPanel.setOpaque(true);
        Border blackLine =
BorderFactory.createLineBorder(Color.black,3);
        Border checkPanelBorder =
BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
        checkPanel.setBorder(checkPanelBorder);

        add( checkPanel, c );
    }
/** Listens to the check boxes. */
@Override
public void itemStateChanged(ItemEvent e) {
    Object source = e.getItemSelectable();

    if (source == plotSpeedometer) {
        renderSpeedometer = plotSpeedometer.isSelected();
        // System.out.println("speedometer select");
    } else if (source == plotTrail) {
        // System.out.println("trail select");
    } else if (source == plotRealTimeData) {
        // System.out.println("real time data select");
    }

    //Now that we know which button was pushed, find out
    //whether it was selected or deselected.
    if (e.getStateChange() == ItemEvent.DESELECTED) {

    }

    //Apply the change to the string.
    //choices.setCharAt(index, c);
}
@Override
public void actionPerformed(ActionEvent e) {
    JComboBox cb = (JComboBox)e.getSource();
    Color selectedName = (Color)cb.getSelectedItem();
    // System.out.println("Event: "+e+ " Source: "+cb);
    cb.setBackground(selectedName);
}
}

private class ControllerSettings extends JPanel implements
ChangeListener {

```

```

private boolean controllerEnabled;
private double sensitivity;
private double deadzone;
private double steeringGearReduction;
private boolean forceFeedbackEnabled;
private float XDZ, YDZ, ZDZ;

static final int SENS_MIN = 0;
static final int SENS_MAX = 10;
static final int SENS_INIT = 5; //initial frames per second

private JSlider joystickSensitivity;
private JTextField XDZtext, YDZtext, ZDZtext;
private JComboBox<String> comPortSelect;

public ControllerSettings() {
    setBackground(SimData.CalPoly_CREAM);
    setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();
    c.fill = GridBagConstraints.CENTER;

    joystickSensitivity = new JSlider(JSlider.HORIZONTAL,
SENS_MIN, SENS_MAX, SENS_INIT);
    joystickSensitivity.addChangeListener( this );
    joystickSensitivity.setMajorTickSpacing( 1 );
    joystickSensitivity.setPaintTicks(true);
    //Create the label table.
    Hashtable<Integer, JLabel> labelTable = new
Hashtable<Integer, JLabel>();
    //PENDING: could use images, but we don't have any good
ones.
    labelTable.put(new Integer( 0 ), new JLabel("Stop") );
    //new JLabel(createImageIcon("images/stop.gif")) );
    labelTable.put(new Integer( SENS_MIN/10 ), new
JLabel("Sensitive") );
    //new JLabel(createImageIcon("images/slow.gif")) );
    labelTable.put(new Integer( SENS_MAX ), new
JLabel("Not Sensitive") );
    //new JLabel(createImageIcon("images/fast.gif")) );
    joystickSensitivity.setLabelTable(labelTable);

    XDZtext = new JTextField();
    XDZtext.setText("0.0");
    YDZtext = new JTextField();
    YDZtext.setText("0.0");

```

```

ZDZtext = new JTextField();
ZDZtext.setText("0.0");

JPanel checkPanel = new JPanel( new GridLayout(0,1) );
//checkPanel.setBackground(SimData.CalPoly_CREAM);
checkPanel.add( new JLabel("Joystick Settings") );
checkPanel.add( new JLabel("Steering Sensitivity") );
checkPanel.add( joystickSensitivity );
checkPanel.add( new JLabel("Set Throttle (X-Axis) dead
zone") );
checkPanel.add( XDZtext );
checkPanel.add( new JLabel("Set Brake (Y-Axis) dead
zone") );
checkPanel.add( YDZtext );
checkPanel.add( new JLabel("Set Steer (Z-Axis) dead
zone") );
checkPanel.add(ZDZtext );
checkPanel.add( new JLabel("Keyboard Settings") );
Border blackLine =
BorderFactory.createLineBorder(Color.black,3);
Border checkPanelBorder =
BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
checkPanel.setBorder(checkPanelBorder);

add( checkPanel, c );
}

/** Listen to the slider. */
@Override
public void stateChanged(ChangeEvent e) {
    JSlider source = (JSlider)e.getSource();
    if (!source.getValueIsAdjusting()) {
        int sensitivity = (int)source.getValue();
        if (sensitivity == 0) {
            ;
        } else {
            ;
        }
    }
}

private class SimulationSettings extends JPanel implements
ActionListener {
    private int FPSlimit;
    private boolean limitFPS;
    private String recordFilePath;
}

```

```

private boolean enableSoundEffects;
private boolean recordData;
private boolean recordCarData;
private boolean recordObstacleData;
// private boolean record
private boolean useSimulatedVelocity; //or used dyno velocity
with simulated side slip to determine velocity components

private JTextField filePath;
private JPanel buttonPanel;
private JButton setFileButton;

private float lidarNoise;

// private JFileChooser fileChooser;

public SimulationSettings() {
    limitFPS = false;
    enableSoundEffects = true;

    setBackground(SimData.CalPoly_CREAM);

    GridLayout buttonLayout = new GridLayout(1,0);
    //Set up the horizontal gap value
    buttonLayout.setHgap(25);
    buttonPanel = new JPanel();
    buttonPanel.setBackground(SimData.CalPoly_CREAM);

    buttonPanel.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    // buttonPanel.setSize(new Dimension(300,180));
    buttonPanel.setLayout(buttonLayout);
    createSetFileButton();
    buttonPanel.add(setFileButton);

    add(buttonPanel);
}

@Override
public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();
    // System.out.println(e.getSource());
    // System.out.println(action);
    if(action.equals("File")) {
        createJFileChooser();
    }
}
private void createJFileChooser() {

```

```

final JFileChooser fileChooser = new JFileChooser();
ActionListener fileListener = new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        //Process action performed
        String action = (String)
ae.getActionCommand());
        // System.out.println( action );
        if( action == "ApproveSelection") { }
        // System.out.println( (String)
fileChooser.getSelectedFile().getAbsolutePath() );

        remove( fileChooser );
    }
};

fileChooser.addActionListener( fileListener );
add( fileChooser );
validate();
}

private void createSetFileButton() {
    setFileButton = new JButton("FILE...");

    setFileButton.setVerticalTextPosition(AbstractButton.CENTER);

    setFileButton.setHorizontalTextPosition(AbstractButton.CENTER);
    setFileButton.setMnemonic('s');
    setFileButton.setActionCommand("File");
    setFileButton.addActionListener(this);
}

/**
 * \class EmbeddedSettings
 * \brief EmbeddedSettings class is responsible for handling and passing
all Embedded Setting GUI parameters
 *
 * to the simulation
 */
public class EmbeddedSettings extends JPanel implements ActionListener
{

    private String COMPORT = "";
    private JComboBox<String> comPortSelect;

    public EmbeddedSettings() {
        //setLayout(new BorderLayout());
        setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.CENTER;

```

```

        setBackground(SimData.CalPoly_CREAM);
        String[] serialPorts =
SerialCommunicationJSSC.getSerialPorts();
        // try {
            // COMPORT = serialPorts[0];
        // } catch(Exception except) {}
        comPortSelect = new JComboBox<String>(serialPorts);
        comPortSelect.insertItemAt("",0);
        JPanel comPortSelectionPanel = new JPanel();
        comPortSelectionPanel.setLayout(new GridLayout(0,1));
        comPortSelectionPanel.add( new JLabel("Embedded
Device Settings") );
        comPortSelectionPanel.add( new JLabel("Serial COM Port
Selection:") );
        comPortSelectionPanel.add( comPortSelect );
        Border blackLine =
BorderFactory.createLineBorder(Color.black,3);
        Border comPortPanelBorder =
BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
        comPortSelectionPanel.setBorder(comPortPanelBorder);

        add(comPortSelectionPanel, c);

        comPortSelect.addActionListener( this );
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String portName = (String)cb.getSelectedItem();
        // System.out.println("Selected port: "+portName);
        COMPORT = portName;
    }
    public String getSerialPort() {
        return COMPORT;
    }
}
private class ColorComboBoxEditor extends JButton implements
ListCellRenderer {
    protected DefaultListCellRenderer defaultRenderer = new
DefaultListCellRenderer();

    public ColorComboBoxEditor() {
        setOpaque(true);
    }
}

```

```

        public Component getListCellRendererComponent(JList list,
Object value, int index, boolean isSelected, boolean cellHasFocus) {
            JLabel renderer = (JLabel)
defaultRenderer.getListCellRendererComponent(list, value, index, isSelected,
cellHasFocus);
            if (value instanceof Color) {
                renderer.setBackground((Color) value);
            }
            return renderer;
        }
    }
}

```

vdhils/SimData.java:

```

/*
 *      \file SimData.java
 *      \brief Represents the data or data interface of the application to draw and update.
 *
 *      Revisions:
 *
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
```

```

*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****



package vdhils;
/// \todo Clean up imports
/// \todo implment speed from encoder boolean to switch between simulating speed and
measuring speed
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Random; // Provides puesdo-random generator
import java.util.concurrent.*; // Provides thread concurrency
import java.io.*; // Provides serializable
import java.util.logging.FileHandler;
import java.util.logging.Formatter;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
import java.util.*;
import java.text.*;
// Joystick API Imports
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;

/**  \class SimData
 *   \brief Handles, stores, shares the application data.
 *
 */

```

```

public class SimData implements Serializable {
    /** Singleton instance of SimData class in order to comply with singleton java
    design pattern */
    private static SimData simData = null;
    private final static Dimension screenSize =
    Toolkit.getDefaultToolkit().getScreenSize();
    /** Serializable UID for saving/loading functionality */
    private static final long serialVersionUID = 7989078461104748964L;
    /** Variable to hold the SettingsData of the application and simulation including
    graphics settings,
        *      controller settings, embedded hardware settings, etc...
        */
    private SettingsData settingsData = new SettingsData();
    public static final Color CalPoly_GOLD = new Color(122,91,17);
    public static final Color CalPoly_GREEN = new Color(2,73,48);
    public static final Color CalPoly_CREAM = new Color(214,204,175);
    /** Scale in pixels per meter (pixels/m) */
    public static float scale = 10.0f;
    /** True if the car's data shall be recorded to a file at the start of each update */
    public static boolean recordCarData = true;
    /** True if the obstacle's data shall be recorded to a file at the start of each update
    */
    public static boolean recordObstacleData = true;
    /** True if the simulation is currently "running", i.e. the sim loop is looping */
    private static boolean running = false;
    /** True if the simulation is currently "running", i.e. the sim loop is looping */
    private static boolean paused = true;
    /** True if the simulation is under joystick control */
    private static boolean joystickControl = false;
    /** True if joystick has force feedback capabilities */
    private static boolean isJoystickFF = false;
    /** Milliseconds between updates when not running the sim at max speed */
    private static final long slowWaitTimeBetweenUpdates = 100;
    /** Milliseconds between updates when running the sim at max speed,
    in this case, no waiting at all with a time of 0 */
    private static final long fastWaitTimeBetweenUpdates = 20;
    /** Represents the known bounds of the sim's area */
    /** Milliseconds between updates when not running the sim at max speed */
    private static final long slowWaitTimeBetweenRenders = 20;
    /** Milliseconds between updates when running the sim at max speed,
    in this case, no waiting at all with a time of 0 */
    private static final long fastWaitTimeBetweenRenders = 0;
    /** Represents the known bounds of the sim's area */
    private Rectangle boundedSimArea;
    /** VDHILS Simulation Object */
    private HardwareInLoopSimulation simulation;
}

```

```

    /** Holds the simulation run time */
    private static double simTime;
    /** Delay controlling the update speed */
    private long currentUpdateSpeed;
    /** Delay controlling the update speed */
    private long currentRenderSpeed;
    /** Previous Time */
    private long oldTimeUpdate,oldTimeRender;
    /** Holds the elasped time FPS was calculated */
    private long timeSinceLastFPSCalculation = 0;
    /** Holds the elasped time UPS was calculated */
    private long timeSinceLastUPSCalculation = 0;
    /** The number of frames */
    private int frames = 0;
    /** The number of simulation updates */
    private int updates = 0;
    /** Holds the latest calculated value of frames per second */
    private int fps = 0;
    /** Holds the latest calculated value of updates per second */
    private int ups = 0;
    /** Holds reference to the joystick */
    private transient static FFJoystick joystick;
    /** Joystick effect */
    private static Effect eff;
    /** VDHILS Logger */
    public static final Logger logger = Logger.getLogger("VDHILS LOG");
    /** Filehandler for the saving/loading of serialized object operations */
    private static FileHandler fh;
    /** Holds reference to input enabled flag */
    protected static boolean inputEnabled = true;

    /**
     * Constructs a new SimData. SimData's constructor creates and
     * schedules an initial updating task
     *
     * \param width The width of the bounded sim area
     * \param height The height of the bounded sim area
     */
    public SimData(int width, int height) {
        simTime = 0.0;
        // Define the know simulation area
        boundedSimArea = new Rectangle(width, height);
        // Initialize random object to obtain psuedo-random numbers
        //Random random = new Random();
        try {
            Date dateNow = new Date();

```

```

        SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
        fh = new FileHandler("./logs/LogFile-
"+dateNowFormater.format(dateNow)+".log");
        logger.addHandler(fh);
        SimpleFormatter formatter = new SimpleFormatter();
        fh.setFormatter(formatter);
        logger.setUseParentHandlers(false);
        logger.info("LOGFILE CREATED");
    } catch(SecurityException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    }

        // Initialize the time, fps, and other variables
currentUpdateSpeed = 5;
        // currentUpdateSpeed = fastWaitTimeBetweenUpdates;
currentRenderSpeed = 50;
        // currentRenderSpeed = fastWaitTimeBetweenRenders;
        // Initialize other parameters explicitly
oldTimeUpdate = System.nanoTime();
        oldTimeRender = System.nanoTime();
}

    public synchronized void writeDataToFile(String filepath) {
        try {
            // Serialize data object to a file
            ObjectOutputStream DataOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
            DataOut.writeObject(this);
            DataOut.close();

            //Serialize data object to a byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            DataOut = new ObjectOutputStream(bos);
            DataOut.writeObject(this);
            DataOut.close();
            byte[] buf = bos.toByteArray();
            SimData.logger.info("Save data successfully");
        } catch (IOException e) {
            SimData.logger.warning("Failed to write data object to file (" +
e.toString() + ")");
        }
    }

/**

```

```

* Resets the last known time of the last update,
* useful for when there has not been an update in a long time,
* or when the application first starts where there may have been
* some time the application spent doing other things between when
* updating starts and the elaboration of the object.
*/
public void resetTimeOfLastUpdate() {
    oldTimeUpdate = System.nanoTime();
}
    public void resetTimeOfLastRender() {
    oldTimeRender = System.nanoTime();
}
/** Set reference to VDHILS Simulation object
*/
public void setSimulation(HardwareInLoopSimulation sim) {
    simulation = sim;
}
public void closeJoystick() {
    try{
        if(isJoystickFF) {
            eff.setStrength(0);
            joystick.stopEffect(eff);
            joystick.destroyEffect(eff);
        }
        joystick.stopAll();
        joystick.destroyAll();
        joystick = null;
    } catch(Exception JoystickNull) { logger.warning("No Joystick to
Close");}
}
public synchronized void setJoystick(FFJoystick joystick) {
    this.joystick = joystick;
    eff = joystick.getSimpleEffect();
    eff.setEffectLength(110);
    eff.setStrength(0);
    joystick.newEffect(eff);
    joystick.playEffect(eff,joystick.INFINITE_TIMES);
    settingsData.setJoystick(this.joystick);
}
/** Sets the simulation runtime (from class performing numerical integration)
*/
public void setSimTime(double simulationTime) {
    simTime = simulationTime;
}
/** \return Returns the latest calculated updates per second */
public static double getSimTime() {

```

```

        return simTime;
    }
    /**
     * Retrieves the time between updates to wait
     */
    public long getCurrentWaitTimeBetweenUpdates() {
        return currentUpdateSpeed;
    }
    /**
     * Retrieves the time between updates to wait
     */
    public long getCurrentWaitTimeBetweenRenders() {
        return currentRenderSpeed;
    }
    public void setJoystickControl(boolean joystickControl) {
        this.joystickControl = joystickControl;
    }
    public void setJoystickFF(boolean isJoystickFF) {
        this.isJoystickFF = isJoystickFF;
    }
    public boolean getJoystickControl() {
        return joystickControl;
    }
    /**
     * \return Returns the latest calculated updates per second */
    public int getUPS() {
        return ups;
    }
    /**
     * Set the frames per second */
    public void setFPS(int fps) {
        this.fps = fps;
    }
    /**
     * \return Returns the latest calculated frames per second */
    public int getFPS() {
        return fps;
    }
    /**
     * Start running the sim data or signal to pause */
    public void setRunning(boolean ToRunOrNotToRun) {
        running = ToRunOrNotToRun;
    }
    /**
     * get paused signal */
    public void setPaused(boolean paused) {
        if(paused)
            inputEnabled = false;
        else
            inputEnabled = true;
        this.paused = paused;
    }
}

```

```

public boolean getRunning() {
    return running;
}
/** Get paused signal */
public boolean getPaused() {
    return paused;
}
/** Updates any objects that need to know how much time has elapsed to update any
needed movements, animations, or events. */
public synchronized void update() {
    // Calculating a new fps/ups value every second
    if (timeSinceLastUPSCalculation >= 1000000000) {
        ups = updates;
        timeSinceLastUPSCalculation = timeSinceLastUPSCalculation -
1000000000;
        updates = 0;
    }
    // Read steering wheel input
    String buttons = "";
    float acceleration = 0;
    float steerangle = 0;
    float brakes = 0;

    if((running == true)&&(paused==false)) {
        if(joystickControl == true) {
            try {
                // Poll joystick for input
                joystick.poll();
                /// \todo Enable user mapping of joystick buttons for
hotkeys (i.e. toggle speedometer, real time plotter, ...)
                if (joystick.isButtonPressed(0) &&
joystick.isButtonPressed(1)) {
                    System.out.println("Buttons 0 and 1 were
pressed at the same time on\njoystick "+joystick.getIndex()+".
Closing program.");
                    //running = false;
                }
                /// \todo cruise control mode - sets throttle set point
so that current speed is maintained
                if(joystick.isButtonPressed(2)) {
                    }
                // Read in joystick poll into appropriate variables
                float f=0;
                for (int a = 0; a < joystick.getAxisCount(); a++) {
                    f = joystick.getAxisValue(a)*100;
                    if(a==0) {

```

```

                acceleration = ((100-f)/200)*100;
            }
        else if(a==1) {
            brakes = ((100-f)/200)*100;
        }
        else if(a==3) {
            steerangle =
(float)(f*(Math.PI/180));
        }
    }
// Set user's car steering and throttle with user's
joystick input
simulation.forwardCarThrottle(acceleration);
simulation.forwardBrakes(brakes);
simulation.forwardSteerangle(steerangle);
// Set Steering wheel torque
if(eff != null) {
    // Linear torque vs. slip angle (linear tire
region)
    eff.setStrength((int)(-
steerangle*(32767/100) ) );
    joystick.updateEffect(eff);
}
} catch(Exception JoystickUpdate) {
logger.warning("Joystick Update Failed");
}

long elapsedTimeUpdate = System.nanoTime() - oldTimeUpdate;
double elapsedTimeSecsUPS =
elapsedTimeUpdate/1000000000.0;

// Update the simulation if the sim is running. (Perform numerical
integration to get the car states at the next timestep).
simTime = simTime + elapsedTimeSecsUPS;
simulation.update(elapsedTimeSecsUPS);
// simulation.update(elapsedTime,acceleration,steerangle,brakes);
oldTimeUpdate = oldTimeUpdate + elapsedTimeUpdate;
timeSinceLastUPSCalculation = timeSinceLastUPSCalculation +
elapsedTimeUpdate;
// An update occurred, increment UPS counter.
updates++;
}
}

/** Draws the SimData as needed */

```

```

public synchronized void drawSimData(Graphics2D drawingBoard, int
drawAreaWidth, int drawAreaHeight) {
    // Calculating a new fps/ups value every second
    if (timeSinceLastFPSCalculation >= 1000000000) {
        fps = frames;
        timeSinceLastFPSCalculation = timeSinceLastFPSCalculation - 1000000000;
        frames = 0;
    }
    // This allows our text and graphics to be nice and smooth
    drawingBoard.setRenderingHint(
        RenderingHints.KEY_TEXT_ANTIALIASING,
        RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    drawingBoard.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    // Always draw over the image with a blank background, so we
    // don't see the last frame's drawings!
    drawingBoard.setColor(Color.LIGHT_GRAY);
    drawingBoard.fillRect(0, 0, drawAreaWidth, drawAreaHeight);
    // Creating a graphics object to not clobber parameter drawingBoard
    // where Car's drawing method may change some state of
    // the drawingBoard parameter graphics object
    Graphics simGraphics = drawingBoard.create();

    if(running == true) {
        simulation.draw(simGraphics);
    }
    // Dispose of the graphics object as were done drawing to it
    simGraphics.dispose();

    long elapsedTimeRender = System.nanoTime() - oldTimeRender;
    double elapsedTimeSecsFPS = elapsedTimeRender/1000000000.0;

    oldTimeRender = oldTimeRender + elapsedTimeRender;
    timeSinceLastFPSCalculation = timeSinceLastFPSCalculation +
elapsedTimeRender;
    // Increment the number of rendered frames
    frames++;
}
public SettingsData getSettingsData() {
    return settingsData;
}
}

vdhils/SimUpdater.java:
```

/\*\*

```

*      \file SimUpdater.java
*      \brief SimUpdater handles the high level updating of the sim,
*             as well as handles updating time based events.
*
*      References:
*
*      Revisions:
*          \li 4/6/2013 TFS
*
*      License:
*      This file is copyright 2014 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*********************************************************************
*****
```

```

package vdhils;
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
```

```

import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, and other utilities)
import java.util.Random;
import java.util.concurrent.*;
import java.util.*;
import java.text.*;
/**  \class SimUpdater
 *   \brief SimUpdater handles the high level rendering of the sim at the
BufferStrategy level,
*          as well as handles updating time rendering based events.
*/
class SimUpdater {
    /** Static reference to instance of simulation data */
    private static SimData simData;

    public SimUpdater(SimData simData) {
        // Set reference
        this.simData = simData;
    }
    /**
     * Simulation update loop run in a separate thread
     */
    public void updateLoop() {
        simData.resetTimeOfLastUpdate();
        // Just loop and loop forever updating the simulation state
        while (true) {
            long nanoTimeAtStartOfUpdate = System.nanoTime();
            // If enough time has passed, update the simulation and all its data
            simData.update();
            // Wait until next update
            waitUntilNextUpdate(nanoTimeAtStartOfUpdate);
        }
    }
}

/**
 * Sleeps the current thread if there's still sometime the application
 * can wait for until the time the next update is needed.
 *
 * \param nanoTimeCurrentUpdateStartedOn Time that current update
 *                                         started
 */
private void waitUntilNextUpdate(long nanoTimeCurrentUpdateStartedOn) {

```

```
// Only sleep to maintain the update speed if speed is higher than
// zero, because Thread.sleep(0) is not defined on what that
// exactly does
long currentUpdateSpeed = simData.getCurrentWaitTimeBetweenUpdates();
if (currentUpdateSpeed > 0) {
    // This could be more accurate by sleeping what's needed on
    // average for the past few seconds
    long timeToSleep = currentUpdateSpeed - ((System.nanoTime() -
nanoTimeCurrentUpdateStartedOn) / 10000000L);
    // If the speed of updating was so slow that it's time for
    // the next update, then choose 0
    timeToSleep = Math.max(timeToSleep, 0);
    // Again, avoiding Thread.sleep(0)
    if (timeToSleep > 0) {
        try {
            Thread.sleep(timeToSleep);
        } catch (InterruptedException e) {
            // It's okay if we're interrupted, program will just run
            // faster.
            Thread.currentThread().interrupt();
        }
    }
}
}
```

## vdhils/Updatable.java:

```
package vdhils;  
  
public interface Updatable {  
    public void update(double elaspedTime);  
}
```

## vdhils/Recordable.java:

```
package vdhils;  
  
public interface Recordable {  
    public void record();  
}
```

vdhils/Vehicle/Car.java:

```
/**  
 *      \file Car.java
```

```

*   \brief Car class with embedded state equations, Car ("bike") model parameters,
*   driver model, and steering/throttle controllers.
*
*   References:
*
*   Revisions:
*
*   License:
* This file is copyright 2014 by T Stevens and released under the Lesser GNU
* Public License, version 2. It intended for educational use only, but its use
* is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

/// \todo Add wheel class to handle skid graphics and 4 instances of wheel with steering angle for front, etc...

/// \todo If no embedded target use build in java classes

/// \todo Implement different sensors (like radar/ultrasonic, magnetic, and "ideal/omnipotent"

/// \todo Draw wheel trials

/// \todo Allow user to specify FWD, RWD, AWD from UI and use appropriate math model

/// \todo Implement locked wheel/tire physics ( locked tire lat force =  $\mu_s * F_z * \sin(\text{slipangle})$  )

/// \todo Extend Model to 3D to allow 3D graphics in future versions

package vdhils.Vehicle;

```

// Import Libraries
import vdhils.*;
import vdhils.SimData;
import vdhils.Graphics.*;
// GUI Imports
import java.awt.Dimension;
import java.awt.*;
import java.awt.FontMetrics;
import java.awt.event.*;
import java.awt.event.KeyEvent;
import java.awt.geom.Line2D;
import java.awt.geom.*;
import java.awt.Toolkit;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.Timer;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Arrays;
import java.util.Random;
import java.util.*;
import java.text.*;
import java.util.ArrayList;
import static java.lang.Math.*;
// Serializable Import, File Operation Imports
import java.io.*;
import static java.nio.file.StandardOpenOption.*;
import java.nio.file.*;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
// Geometry Library Import
import GeometryUtil.*;

/**
 * \class Car
 * \brief Car class with embedded state equations, car model parameters
 *
 * The car model class contains all the parameters for the bicycle dynamic
model
* as well as the state equations used to step forward in time using a Euler method
* integration scheme. This class handles not only all of the physics associated with
the
* care class but also handles the rendering of the car's wireframe, textures, forces,
* data, etc... This car class is capable of simulating a LiDAR sensor that can be
placed

```

```

*      anywhere on the car and communicating with an embedded system's computer for
segmentation
*      path-planning purposes to override the user's control inputs with safe control
inputs if
*      the system signals a high-risk situation or imminent collision.
*/
public class Car extends Sprite implements Serializable, PlottableSignals, Recordable {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
serialized output stream */
    private static final long serialVersionUID = 7989078461104748963L;

/*! Enumerator to signify whether vehicle if front or rear wheel drive. NOTE: To
be implemented in future revisions */
public static enum DRIVE {
    RWD, /*! RWD = Rear wheel drive */
    FWD      /*! FWD = Front wheel drive */
};
/*! Boolean indicating whether vehicle speed (motor) should be measured from
encoders or simulated*/
private boolean updateSpeedFromEncoders = false;
/** Beaglebone embedded linux hardware represented as SerialCommunication
object that will handle all reading and writing
* to the Beaglebone via UART serial communication
*/
private SerialCommunicationJSSC embeddedHardware = null;

protected Thread processingThread;
///< Thread that sends buffered data to embedded hardware when started
private boolean AIControlled;
///< Boolean to indicate whether car is under user control or AI control (typically
only one user car will be present in each simulation)
protected boolean processData = false;
///< Boolean to indicate if the Beaglebone embedded hardware is currently
processing data (a point-cloud)
private float LOW_SPEED_THRES = 0.0f;
///< Speed at which the low speed Ackermann steer model is used and slip angles
are assumed to be negligible (Lowest possible value with smooth model transition and no
instabilities is the goal)

/// and without observed low-speed instability at max steer
angle)
private TireModel tireModel;

```

```

public PIDcontroller
PID throttle controller

    /// \todo Replace tiremodel with front and rear tire model
    // private TireModel           speedController = null;           ///  

    // private TireModel           frontTireModel;
    // private TireModel           rearTireModel;
    private MotorModel           motorModel;
    ///< Electric Motor Model from Control Theory
    private LiDAR                lidar;

        ///< Simulated LiDAR sensor
    private boolean
    private boolean
    float[][]

    private float
    private float
    ///< Drag coefficient
    private float
    ///< Rolling resistance coefficient
    private float
    ///< Mechanically imposed steering limit
    private float
    ///< Car mass (kg)
    private float
    ///< Car inertia (kg*m^2)
    private float
    ///< Car wheel mass as percentage of total mass (0-100%)
    private float
    ///< Car wheel inertia (kg*m^2)
    private float
    ///< Distance from CG to front tire (m)
    private float
    ///< Distance from CG to rear tire (m)
    private float
    ///< Distance from CG to ground tire (m)
    private float
    ///< Car wheelbase distance (m)
    private float
    ///< Car length (m)
    private float
    ///< Car width (m)
    private float
    Car wheel's length (m)
        private float
    ///< Car wheel's width (m)
    protected float
    ///< Look ahead distance

```

```

protected ArrayList<Line2D.Float> path;
///< Pure-pursuit path follower - path to follow
protected float batteryCapacity = 3300f; ///<
Car battery capacity (mAh rating)
protected float powerTrainEff = 1.0f; ///<
Power-train efficiency
private String Manuever = "None";
private boolean enableManuever;

private boolean detectWheels = true;
private float placeLidar_x = 0.0f;
private float placeLidar_y = 0.0f;// 1.25f;

private double screen_pos_x;
private double screen_pos_y;
private float[][] bodyCorners = new float[4][2];
private float[][] wheels = new float[4][2];
private float[][] wheelCenters = new float[4][2];
// Vehicle State in World/Global Coordinate Frame
private double position_WC_x;
private double position_WC_y;
private double velocity_WC_x;
private double velocity_WC_y;
private double acceleration_WC_x;
private double acceleration_WC_y;
// Vehicle State in Vehicle Coordinate Frame
private double angle;
private double angularvelocity;
private double angular_acceleration;
private double velocity_x;
private double velocity_y;
private double sim_velocity_x;
private double sim_velocity_y;
private double acceleration_x;
private double acceleration_y;
private double rot_angle;
private double rot_angle_front;
private double rot_angle_rear;
private double sideslip;
private double slipanglefront;
private double slipanglerear;
private double force_x;
private double force_y;
private int rear_slide;
private int front_slide;
private double resistance_x;

```

```

private double resistance_y;
private double torque;
private double yawspeed;
private double yawspeed_f;
private double yawspeed_r;
private double weight;
private double ftraction_x;
private double ftraction_y;
private double flatf_x;
private double flatf_y;
private double flatr_x;
private double flatr_y;
private double Nf;
private double Nr;
// Control Signals
private float steerangle;
private float prevsteerangle;
private float steerGearReduct = 12.0f;
private float throttle = 0f;
private float brake;
private float speed;
// Trail Parameters
private int TRAIL_SIZE = 150;
private float[][] trail;
private int num_trail;
// Real Time Plotter Parameters
private final int graphNum = 20;
private int toolBarHeight;
private float[] Vxpts = new float[graphNum];
private float VxPtsMax;
private float[] Vypts = new float[graphNum];
private float VyPtsMax;
private float[] yawRatepts = new float[graphNum];
private float yawRateptsMax;
private float[] SteerAngle = new float[graphNum];
private int realTimePlotterWidth = 300;
private int realTimePlotterHeight = 180;
private boolean interpPlot = true;

private String velocity_x_LEGEND = "Velocity_x";
private String velocity_y_LEGEND = "Velocity_y";
private String speed_LEGEND = "Speed";

private int fslideCount = 0;
private int rslideCount = 0;

```

```

private BufferedWriter recordStream;
private BufferedWriter pointCloud;

private Dimension screen;

public Car(Dimension screen, float spritescale, float xpos, float ypos, float angle,
float radarRange, String carSelect, boolean AIControlled, BufferedWriter recordStream)
{
    super(screen,spritescale,carSelect,true);
    addImage("/Resources/TireTexture1.png",false);
    addImage("/Resources/textures/Tires/TireSkidFading.png",true);
    // carParameters = new CarParameters();
    this.AIControlled = AIControlled;
    this.angle = angle;
    this.recordStream = recordStream;
    this.screen = screen;
    // Initialize a new LiDAR sensor if the car is under user control
    if(!AIControlled)
        lidar = new LiDAR(radarRange);

    // Create new car tire model
    tireModel = new TireModel();
    // Set model property of the tire model to use the "Magic formula" when
determining lateral forces
    tireModel.model = TireModel.MODEL.MAGIC;
    // Create new car motor model
    motorModel = new MotorModel();
    inertia_wheel =
(float)((wheel_mass/100.0f)*mass*pow(wheeldiameter/2,2));      // model wheel as
hoop
    // Initialize vehicle state
    angularvelocity = 0;
    position_WC_x = xpos;
    position_WC_y = ypos;
    velocity_WC_x = 0;
    velocity_WC_y = 0;
    // Initialize vehicle inputs
    steerangle = 0;
    throttle = 0;
    brake = 0;
    // Initialize weight and static normal forces
    weight = mass*9.81f;
    Nf = (weight*R2CG)/wheelbase;
    Nr = weight-Nf;

    trail = new float[ TRAIL_SIZE ][3];
}

```

```

        num_trail = 0;

        if(!AIControlled) {
            embeddedHardware = new SerialCommunicationJSSC();
            embeddedHardware.connect();
        }
        try {
            if(!AIControlled)
                recordStream.write("Time SteerAngle(rad) Brake(%)
Throttle(%) WC_x(m) WC_y(m) SideSlip(rad) Angle(rad) Vx(m/s) Vy(m/s)
YawRate(rad/s) Threat_ECU");
            else
                recordStream.write("Time SteerAngle(rad) Brake(%)
Throttle(%) WC_x(m) WC_y(m) SideSlip(rad) Angle(rad) Vx(m/s) Vy(m/s)
YawRate(rad/s)");
            recordStream.newLine();
            recordStream.flush();
        } catch( IOException x ) {
            SimData.logger.warning("Failed to write header to car data record
file (""+x+"")");
        } catch( Exception otherExceptions ) {
            SimData.logger.warning("Failed to write to car data record file
("")+otherExceptions+"");
        }
    }

    Date dateNow = new Date(); // Create a
unique & formatted date/time based filename
    SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
    String formatDateNow = dateNowFormater.format(dateNow).toString();
    String carRecordFP = "Car_PtCloud_" +formatDateNow + ".txt";
                    // Car record stream filename
    Path carPath = FileSystems.getDefault().getPath("Recorded
Data",carRecordFP);
    try {
        pointCloud = new BufferedWriter(new
FileWriter(carPath.toFile().getAbsoluteFile())); // Initialize Car record output stream
        pointCloud.write("Time X_Pts(m) Y_Pts(m)");
        pointCloud.newLine();
        pointCloud.flush();
        SimData.logger.info("Successfully created car pt cloud record
file");
    } catch( IOException x ) {
        SimData.logger.warning("Failed to create car pt cloud record file
("")");
    }
}

```

```

        } catch(Exception otherExceptions) {
            SimData.logger.warning("Failed to create car pt cloud record file
(" + otherExceptions + ")");
        }
    }

    public Car( Dimension screen, float spritescale, CarParameters carParameters,
boolean AIControlled, BufferedWriter recordStream) {
        super(screen,spritescale,carParameters.getCarImage(),true);
        addImage("/Resources/TireTexture1.png",false);
        addImage("/Resources/textures/Tires/TireSkidFading.png",true);
        this.AIControlled = AIControlled;
        this.angle = carParameters.getAngle();
        this.recordStream = recordStream;
        this.screen = screen;

        mass = carParameters.getMass();
        inertia = carParameters.getInertia();
        F2CG = carParameters.getF2CG();
        R2CG = carParameters.getR2CG();
        h = carParameters.getH();
        length = carParameters.getLength();
        width = carParameters.getWidth();

        STEER_LIMIT = carParameters.getSteerlimit();
        wheeldiameter = carParameters.getWheeldiameter();
        steerGearReduct = carParameters.getSteerGearReduct();

        LOW_SPEED_THRES = carParameters.getLowspeedthres();
        DRAG = carParameters.getDrag();
        fr = carParameters.getResistance();

        TRAIL_SIZE = carParameters.getTrailLength();

        if(!AIControlled) {
            updateSpeedFromEncoders =
carParameters.getUpdateSpeedfromEncoders();
            lidar = new LiDAR(carParameters.getSensorRange(),
carParameters.getScanAngle(),
carParameters.getSensorFrequency(),carParameters.getNumberOfBeams());
        }
        else
            updateSpeedFromEncoders = false;

        // Create new car tire model
        if(carParameters.isTireModelLinear()) {

```

```

        tireModel = new TireModel(carParameters.getCA_R(),
carParameters.getCA_F());
    } else {
        tireModel = new TireModel(carParameters.getB(),
carParameters.getC(), carParameters.getD(), carParameters.getE());
    }
    tireModel.setStaticFrictionCoeff(carParameters.getMew());
    tireModel.setDynamicFactorCoeff(carParameters.getK());

    // Create new car motor model
    motorModel = new
MotorModel(wheeldiameter/2,carParameters.getRa(),carParameters.getKt(),carParameter
s.getKv(),carParameters.getEa(),carParameters.getGearratio());
        // Model the wheel as a hoop
        inertia_wheel =
(float)((wheel_mass/100.0f)*mass*pow(wheeldiameter/2,2));

        // Initialize vehicle state
        angularvelocity = 0;
        position_WC_x = carParameters.getXPos();
        position_WC_y = carParameters.getYPos();
        velocity_WC_x = 0;
        velocity_WC_y = 0;

        steerangle = 0;
        throttle = 0;
        brake = 0;

        weight = mass*9.81f;
        Nf = (weight*R2CG)/wheelbase;
        Nr = weight-Nf;

        trail = new float[ TRAIL_SIZE ][3];
        num_trail = 0;

        speedController = new
PIDcontroller(carParameters.getProportional(),carParameters.getIntegral(),carParameters.
getDerivative());
        setSpeedControlSP(carParameters.getSCSP());
        enableSpeedControl(carParameters.getEnableSC());
        if(carParameters.getEnableSC())
            speedController.setLastEnable(-1);

        Manuever = carParameters.getManuever();
        lfw = carParameters.getLFW();

```

```

enableManuever = carParameters.getEnableManuever();

if(!AIControlled) {
    embeddedHardware = new
SerialCommunicationJSSC(carParameters.getSerialPort());
    embeddedHardware.connect();
}

try {
    if(!AIControlled)
        recordStream.write("Time SteerAngle(rad) Brake(%)
Throttle(%) WC_x(m) WC_y(m) SideSlip(rad) Angle(rad) Vx(m/s) Vy(m/s)
YawRate(rad/s) Threat_ECU");
    else
        recordStream.write("Time SteerAngle(rad) Brake(%)
Throttle(%) WC_x(m) WC_y(m) SideSlip(rad) Angle(rad) Vx(m/s) Vy(m/s)
YawRate(rad/s)");
    recordStream.newLine();
    recordStream.flush();
} catch(IOException x) {
    SimData.logger.warning("Failed to write header to car data record
file (""+x+"")");
} catch(Exception otherExceptions) {
    SimData.logger.warning("Failed to write to car data record file
(""+otherExceptions+"")");
}

Date dateNow = new Date(); // Create a
unique & formatted date/time based filename
SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
String formatDateNow = dateNowFormater.format(dateNow).toString();
String carRecordFP = "Car_PtCloud_"+formatDateNow+".txt";
// Car record stream filename
Path carPath = FileSystems.getDefault().getPath("Recorded
Data",carRecordFP);
try {
    pointCloud = new BufferedWriter(new
FileWriter(carPath.toFile().getAbsoluteFile())); // Initialize Car record output stream
    pointCloud.write("Time X_Pts(m) Y_Pts(m)");
    pointCloud.newLine();
    pointCloud.flush();
    SimData.logger.info("Successfully created car pt cloud record
file");
} catch(IOException x) {

```

```

        SimData.logger.warning("Failed to create car pt cloud record file
        (" + x + "));

    } catch (Exception otherExceptions) {
        SimData.logger.warning("Failed to create car pt cloud record file
        (" + otherExceptions + "));

    }

    public void record() {
        try {
            if (!AIControlled)
                recordStream.write(SimData.getSimTime() +
                    "+steerangle+" "+brake+" "+throttle+" "+position_WC_x+" "+position_WC_y+" +
                    "+sideslip+" "+angle+" "+velocity_x+" "+velocity_y+" "+angularvelocity+" +
                    "+embeddedHardware.getRRT_Enable());
            else
                recordStream.write(SimData.getSimTime() +
                    "+steerangle+" "+brake+" "+throttle+" "+position_WC_x+" "+position_WC_y+" +
                    "+sideslip+" "+angle+" "+velocity_x+" "+velocity_y+" "+angularvelocity);
            recordStream.newLine();
            recordStream.flush();
        } catch (IOException x) {
            SimData.logger.warning("Failed to write header to car data record
file (" + x + "));

        } catch (Exception otherExceptions) {
            SimData.logger.warning("Failed to write to car data record file
(" + otherExceptions + "));

        }
        // if (!AIControlled)
        // lidar.record();
    }

    public void enableSpeedControl(boolean controlSpeed) {
        speedController.setEnable(controlSpeed);
    }
    public void setSpeedControlSP(float newSP) {
        speedController.setSP(newSP);
    }
    public String getManuever() {
        return Manuever;
    }
    public boolean getEnableManuever() {
        return enableManuever;
    }
    public void setPath(ArrayList<Line2D.Float> path) { //Line2D.Float path) {

```

```

        this.path = new ArrayList<Line2D.Float>(path);
    }

    public void setDrawPopup(boolean setPopup) {
        drawPopup = setPopup;
    }

    public void togglePopup() {
        if(drawPopup)
            drawPopup = false;
        else
            drawPopup = true;
    }

    private void drawVehicleCoordAxes(Graphics2D g2d) {
        g2d.setColor(Color.red);
        g2d.translate( screen_pos_x, screen_pos_y );
        g2d.draw(new Line2D.Float(0,0,(float)(2.5*wheelbase*cos(angle-PI/2)*SimData.scale),(float)(2.5*wheelbase*sin(angle-PI/2)*SimData.scale)));
        g2d.draw(new Line2D.Float(0,0,(float)(2.5*wheelbase*cos(angle)*SimData.scale),(float)(2.5*wheelbase*sin(angle)*SimData.scale)));
        g2d.translate( -screen_pos_x, -screen_pos_y );
        drawGeneralPath( Geometry.createArrow ((float) screen_pos_x, (float) screen_pos_y,
                                                (float)(screen_pos_x+(2.5*wheelbase*cos(angle-PI/2)*SimData.scale)), (float)(screen_pos_y+(2.5*wheelbase*sin(angle-PI/2)*SimData.scale)),
                                                17.5f, 0.45f, 0.325f),g2d);
        drawGeneralPath( Geometry.createArrow ((float) screen_pos_x, (float) screen_pos_y,
                                                (float)(screen_pos_x+(2.5*wheelbase*cos(angle)*SimData.scale)), (float)(screen_pos_y+(2.5*wheelbase*sin(angle)*SimData.scale)),
                                                17.5f, 0.45f, 0.325f),g2d);
    }

    private void drawGeneralPath(float[] Points,Graphics2D g2) {
        GeneralPath polygon = new
        GeneralPath(GeneralPath.WIND_EVEN_ODD,Points.length);
        for(int i=0;i<Points.length;) {
            if (i == 0)
                polygon.moveTo(Points[i], Points[i+1]);
            polygon.lineTo(Points[i], Points[i+1]);
            i=i+2;
        }
        polygon.closePath();
    }
}

```

```

        g2.fill(polygon);
    }

    private void drawVehicleForces(Graphics2D g2d) {
        g2d.setColor(Color.yellow);

        //RESULTANT FORCES
        // if(drawResultantForces) {
            if(force_x>0) { // if force_x is greater than 0 draw on rear pointing
                towards car else draw on front pointing towards car
                g2d.draw(new
Line2D.Float((corners[2][0]+corners[3][0])/2,(corners[2][1]+corners[3][1])/2,(corners[2]
[0]+corners[3][0])/2+(float)((force_x/(0.1*SimData.scale))*wheelbase*cos(angle+PI/2)*
SimData.scale),(corners[2][1]+corners[3][1])/2+(float)((force_x/(0.1*SimData.scale))*w
heelbase*sin(angle+PI/2)*SimData.scale)));
                // g2d.draw(new
Line2D.Float(corners[3][0],corners[3][1],corners[3][0]+(float)(2.5*wheelbase*cos(angle
+PI)*SimData.scale),corners[3][1]+(float)(2.5*wheelbase*sin(angle+PI)*SimData.scale)
));
                //g2d.translate( -screen_pos_x, -screen_pos_y );
                drawGeneralPath( Geometry.createArrow (
                    (float)((corners[2][0]+corners[3][0])/2+(2.5*wheelbase*cos(angle+PI/2)*SimData.scale)
),
                    (float)((corners[2][1]+corners[3][1])/2+(2.5*wheelbase*sin(angle+PI/2)*SimData.scale))
,
                    (float) (corners[2][0]+corners[3][0])/2, (float)
(corners[2][1]+corners[3][1])/2,
                    17.5f, 0.45f, 0.325f),g2d);
            } else if(force_x<0) {
                g2d.draw(new
Line2D.Float((corners[0][0]+corners[1][0])/2,(corners[0][1]+corners[1][1])/2,(corners[0]
[0]+corners[1][0])/2+(float)((force_x/(0.1*SimData.scale))*wheelbase*cos(angle+PI/2)*
SimData.scale),(corners[0][1]+corners[1][1])/2+(float)((force_x/(0.1*SimData.scale))*w
heelbase*sin(angle+PI/2)*SimData.scale)));
                // g2d.draw(new
Line2D.Float(corners[3][0],corners[3][1],corners[3][0]+(float)(2.5*wheelbase*cos(angle
+PI)*SimData.scale),corners[3][1]+(float)(2.5*wheelbase*sin(angle+PI)*SimData.scale)
));
                //g2d.translate( -screen_pos_x, -screen_pos_y );
                drawGeneralPath( Geometry.createArrow (
                    (float)((corners[0][0]+corners[1][0])/2+(2.5*wheelbase*cos(angle-
PI/2)*SimData.scale)),
                    (float)((corners[0][1]+corners[1][1])/2+(2.5*wheelbase*sin(angle-PI/2)*SimData.scale)),
                    (float) (corners[0][0]+corners[1][0])/2, (float)
(corners[0][1]+corners[1][1])/2,
                    17.5f, 0.45f, 0.325f),g2d);
            }
        }
    }
}

```

```

        }
        if(force_y>0) {
            g2d.draw(new
Line2D.Float((corners[0][0]+corners[3][0])/2,(corners[0][1]+corners[3][1])/2,(corners[0]
[0]+corners[3][0])/2+(float)((force_y/(0.1*SimData.scale))*wheelbase*cos(angle+PI)*Si
mData.scale),(corners[0][1]+corners[3][1])/2+(float)((force_y/(0.1*SimData.scale))*whe
elbase*sin(angle+PI)*SimData.scale)));
            drawGeneralPath( Geometry.createArrow (
(float)((corners[0][0]+corners[3][0])/2+(2.5*wheelbase*cos(angle+PI)*SimData.scale)),
(float)((corners[0][1]+corners[3][1])/2+(2.5*wheelbase*sin(angle+PI)*SimData.scale)),
(float) (corners[0][0]+corners[3][0])/2, (float)
(corners[0][1]+corners[3][1])/2,
17.5f, 0.45f, 0.325f),g2d);
        } else if(force_y<0) {
            g2d.draw(new
Line2D.Float((corners[1][0]+corners[2][0])/2,(corners[1][1]+corners[2][1])/2,(corners[1]
[0]+corners[2][0])/2+(float)((force_y/(0.1*SimData.scale))*wheelbase*cos(angle+PI)*Si
mData.scale),(corners[1][1]+corners[2][1])/2+(float)((force_y/(0.1*SimData.scale))*whe
elbase*sin(angle+PI)*SimData.scale)));
            drawGeneralPath( Geometry.createArrow (
(float)((corners[1][0]+corners[2][0])/2+(2.5*wheelbase*cos(angle)*SimData.scale)),
(float)((corners[1][1]+corners[2][1])/2+(2.5*wheelbase*sin(angle)*SimData.scale)),
(float) (corners[1][0]+corners[2][0])/2, (float)
(corners[1][1]+corners[2][1])/2,
17.5f, 0.45f, 0.325f),g2d);
        }
        // else if(drawAllForces) {
        if(flatr_y>0) {
            g2d.draw(new
Line2D.Float(wheels[3][0],wheels[3][1],wheels[3][0]+(float)((flatr_y/(0.1*SimData.sca
le))*wheelbase*cos(angle+PI)*SimData.scale),wheels[3][1]+(float)((flatr_y/(0.1*SimDat
a.scale))*wheelbase*sin(angle+PI)*SimData.scale));
            drawGeneralPath( Geometry.createArrow (
(float)(wheels[3][0]+(2.5*wheelbase*cos(angle+PI)*SimData.scale)),
(float)(wheels[3][1]+(2.5*wheelbase*sin(angle+PI)*SimData.scale)),
wheels[3][0], wheels[3][1],
17.5f, 0.45f, 0.325f),g2d);
        } else if(flatr_y<0) {
            g2d.draw(new
Line2D.Float(wheels[2][0],wheels[2][1],wheels[2][0]+(float)((flatr_y/(0.1*SimData.sca
le))*wheelbase*cos(angle+PI)*SimData.scale),wheels[2][1]+(float)((flatr_y/(0.1*SimDat
a.scale))*wheelbase*sin(angle+PI)*SimData.scale));
            drawGeneralPath( Geometry.createArrow (
(float)(wheels[2][0]+(2.5*wheelbase*cos(angle)*SimData.scale)),
(float)(wheels[2][1]+(2.5*wheelbase*sin(angle)*SimData.scale)),
wheels[2][0], wheels[2][1],

```

```

    17.5f, 0.45f, 0.325f),g2d);
}
if(flatr_x>0) { // if force_x is greater than 0 draw on rear pointing towards
car else draw on front pointing towards car
    g2d.draw(new
Line2D.Float(wheels[2][0],wheels[2][1],wheels[2][0]+(float)((flatr_x/(0.1*SimData.scal
e))*wheelbase*cos(angle+PI/2)*SimData.scale),wheels[2][1]+(float)((flatr_x/(0.1*SimD
ata.scale))*wheelbase*sin(angle+PI/2)*SimData.scale)));
    drawGeneralPath( Geometry.createArrow (
(float)(wheels[2][0]+(2.5*wheelbase*cos(angle+PI/2)*SimData.scale)),
(float)(wheels[2][1]+(2.5*wheelbase*sin(angle+PI/2)*SimData.scale)),
        (float) wheels[2][0], (float) wheels[2][1],
    17.5f, 0.45f, 0.325f),g2d);
}

if(flatf_y>0) {
    g2d.draw(new
Line2D.Float(wheels[0][0],wheels[0][1],wheels[0][0]+(float)((flatr_y/(0.1*SimData.scal
e))*wheelbase*cos(angle+steerangle+PI)*SimData.scale),wheels[0][1]+(float)((flatr_y/(0
.1*SimData.scale))*wheelbase*sin(angle+steerangle+PI)*SimData.scale));
    drawGeneralPath( Geometry.createArrow (
(float)(wheels[0][0]+(2.5*wheelbase*cos(angle+steerangle+PI)*SimData.scale)),
(float)(wheels[0][1]+(2.5*wheelbase*sin(angle+steerangle+PI)*SimData.scale)),
        wheels[0][0], wheels[0][1],
    17.5f, 0.45f, 0.325f),g2d);
} else if(flatf_y<0) {
    g2d.draw(new
Line2D.Float(wheels[1][0],wheels[1][1],wheels[1][0]+(float)((flatr_y/(0.1*SimData.scal
e))*wheelbase*cos(angle+steerangle+PI)*SimData.scale),wheels[1][1]+(float)((flatr_y/(0
.1*SimData.scale))*wheelbase*sin(angle+steerangle+PI)*SimData.scale));
    drawGeneralPath( Geometry.createArrow (
(float)(wheels[1][0]+(2.5*wheelbase*cos(angle+steerangle)*SimData.scale)),
(float)(wheels[1][1]+(2.5*wheelbase*sin(angle+steerangle)*SimData.scale)),
        wheels[1][0], wheels[1][1],
    17.5f, 0.45f, 0.325f),g2d);
}
}

protected void drawPopupVehicleState(Graphics2D g2d) {
    Composite c = g2d.getComposite();

    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
0.3f));
    g2d.setColor(Color.black);
    g2d.fillRect( (int)screen_pos_x, (int)screen_pos_y, 145, 40);
    g2d.setComposite(c);
}

```

```

        g2d.setFont( new Font("Arail", Font.BOLD, 12));
        g2d.setColor(Color.white);
        g2d.drawString("X: "+String.format("%.2f (m)", position_WC_x)+" Y:
"+String.format("%.2f
(m)",position_WC_y),(float)(screen_pos_x+5),(float)(screen_pos_y+12));
        g2d.drawString("X-Velocity: "+String.format("%.2f (m/s)",
velocity_x),(float)(screen_pos_x+5),(float)(screen_pos_y+24));
        g2d.drawString("Y-Velocity: "+String.format("%.2f
(m/s)",velocity_y),(float)(screen_pos_x+5),(float)(screen_pos_y+36));
    }
    public LiDAR getLidar() {
        return lidar;
    }
    public void setOffsetX(float offsetX) {
        this.offsetX = offsetX;
    }
    public float getOffsetX() {
        return offsetX;
    }
    public void setOffsetY(float offsetY) {
        this.offsetY = offsetY;
    }
    public float getOffsetY() {
        return offsetY;
    }
    public void draw(Graphics g) {
        // Obtain 2D graphics context
        Graphics2D g2d = (Graphics2D) g;
        // Set graphics object to draw with the color blue
        g2d.setColor(Color.blue);
        // Calculate rotation matrix components used repeatedly below
        double sn = sin(angle);
        double cs = cos(angle);
        // Determine the car's position on the screen by transforming from
        global/world coord. frame (meters) to
        // the graphics/screen coordinate frame (pixels)
        screen_pos_x = (float)(position_WC_x * SimData.scale +
(screenSize.getWidth()/2)+offsetX); // scale and translate to center of screen
        screen_pos_y = (float)(-position_WC_y * SimData.scale +
(float)(screenSize.getHeight()/2)+offsetY); // scale, inverse, and translate to center of
screen

        /// \todo If AI controlled and goes off screen - delete? recycle? ???
        // if(!AIControlled) {
            //if(inside innerrectangle)
                // do nothing

```

```

        // outside or on outerrectangle
        // move to edge of innerrectangle and add distance to a
shared offset variable that updates the road and obstacles coords instead
        // for now

        // while(screen_pos_y < 0)
            // screen_pos_y += screenSize.getHeight();
        // while(screen_pos_y > screenSize.getHeight())
            // screen_pos_y -= screenSize.getHeight();
        // while(screen_pos_x < 0)
            // screen_pos_x += screenSize.getWidth();
        // while(screen_pos_x > screenSize.getWidth())
            // screen_pos_x -= screenSize.getWidth();
        //
int bufferSpace = 100;
if(!AIControlled) {
    if(screen_pos_y < bufferSpace) {
        offsetY = (float)(position_WC_y*SimData.scale-
screenSize.getHeight()/2)+bufferSpace;
        screen_pos_y = (float)(-position_WC_y * SimData.scale +
(screenSize.getHeight()/2)+offsetY);
    } else if(screen_pos_y > screenSize.getHeight()-bufferSpace) {
        offsetY = (float)-((-position_WC_y*SimData.scale-
screenSize.getHeight()/2)+bufferSpace);
        screen_pos_y = (float)(-position_WC_y * SimData.scale +
(screenSize.getHeight()/2)+offsetY);
    }
    if(screen_pos_x < bufferSpace) {
        offsetX = (float)((-position_WC_x*SimData.scale-
screenSize.getWidth()/2)+bufferSpace);
        screen_pos_x = (float)(position_WC_x * SimData.scale +
(screenSize.getWidth()/2)+offsetX);
    } else if(screen_pos_x > screenSize.getWidth()-bufferSpace) {
        offsetX = (float)-((position_WC_x*SimData.scale -
screenSize.getWidth()/2)+bufferSpace);
        screen_pos_x = (float)(position_WC_x * SimData.scale +
(screenSize.getWidth()/2)+offsetX);
    }
} else {
    if((screen_pos_y < 0)||(screen_pos_y >
screenSize.getHeight())||(screen_pos_x < 0)|| (screen_pos_x > screenSize.getWidth())))
        // Offscreen flag for removal?
        // removeFlag = true;
}
}

```

```

int col;
float[][] w = new float[4][2];

corners[0][0] = -width/2;
corners[0][1] = -length/2;

corners[1][0] = width/2;
corners[1][1] = -length/2;

corners[2][0] = width/2;
corners[2][1] = length/2;

corners[3][0] = -width/2;
corners[3][1] = length/2;

for(int i = 0; i <= 3; i++)
{
    w[i][0] = (float)(cs*corners[i][0] - sn*corners[i][1]);
    w[i][1] = (float)(sn*corners[i][0] + cs*corners[i][1]);
    corners[i][0] = w[i][0];
    corners[i][1] = w[i][1];
    bodyCorners[i][0] = corners[i][0];
    bodyCorners[i][1] = corners[i][1];
}
for(int i = 0; i <= 3; i++)
{
    corners[i][0] *= SimData.scale;
    corners[i][1] *= SimData.scale;
    corners[i][0] += screen_pos_x;
    corners[i][1] += screen_pos_y;
}

wheels[0][0] = -width/2;
wheels[0][1] = -F2CG;

wheels[1][0] = width/2;
wheels[1][1] = -F2CG;

wheels[2][0] = width/2;
wheels[2][1] = R2CG;

wheels[3][0] = -width/2;
wheels[3][1] = R2CG;

for(int i = 0; i <= 3; i++)
{

```

```

w[i][0] = (float)(cs*wheels[i][0] - sn*wheels[i][1]);
w[i][1] = (float)(sn*wheels[i][0] + cs*wheels[i][1]);
wheels[i][0] = w[i][0];
wheels[i][1] = w[i][1];

wheelCenters[i][0] = wheels[i][0];
wheelCenters[i][1] = wheels[i][1];

wheels[i][0] *= SimData.scale;
wheels[i][1] *= SimData.scale;
wheels[i][0] += screen_pos_x;
wheels[i][1] += screen_pos_y;
}

draw_trail( g2d );

// if(!drawWheelTexture) {
// start drawWireframe
g2d.draw(new
Line2D.Float(corners[0][0],corners[0][1],corners[1][0],corners[1][1]) );
g2d.draw(new
Line2D.Float(corners[1][0],corners[1][1],corners[2][0],corners[2][1]) );
g2d.draw(new
Line2D.Float(corners[2][0],corners[2][1],corners[3][0],corners[3][1]) );
g2d.draw(new
Line2D.Float(corners[3][0],corners[3][1],corners[0][0],corners[0][1]) );

g2d.draw(new
Line2D.Float(corners[0][0],corners[0][1],corners[2][0],corners[2][1]) );
g2d.draw(new
Line2D.Float(corners[1][0],corners[1][1],corners[3][0],corners[3][1]) );

Color origColor = g2d.getColor();
g2d.setColor( new Color( 15, 20, 20 ) );
draw_wheel( g2d, 0, wheels[0][0], wheels[0][1], front_slide );
draw_wheel( g2d, 1, wheels[1][0], wheels[1][1], front_slide );
draw_wheel( g2d, 2, wheels[2][0], wheels[2][1], rear_slide );
draw_wheel( g2d, 3, wheels[3][0], wheels[3][1], rear_slide );
g2d.setColor( origColor );
// end drawWireframe
}

if((front_slide == 1)||(rear_slide ==1))
    drawSlideTexture(g2d);
else {
    fslideCount--;
    if(fslideCount < 0)

```

```

        fslideCount = 0;
        rslideCount--;
        if(rslideCount < 0)
            rslideCount = 0;
    }

    drawWheelTexture( g2d );
    if(!AIControlled)
        lidar.drawPointCloud( g2d );
    // if(drawCarTexture)
        drawCarTexture( g2d );
    //if(skidding)
        // drawSkidTexture( g2d );

    drawVehicleCoordAxes(g2d);
    drawVehicleForces(g2d);

    if(!AIControlled) {
        drawRealTimePlotter( g2d, (int)(screenSize.getWidth()-
realTimePlotterWidth), toolBarHeight );
    }
    if(drawPopup)
        drawPopupVehicleState( g2d );
}

public Rectangle2D.Float getBounds2D() {
    float maxLength = max(max(abs(corners[1][0] - corners[0][0]),
abs(corners[1][1] - corners[0][1])), abs(corners[2][1]-corners[1][1]));
    return new Rectangle2D.Float(corners[0][0], corners[0][1], maxLength,
maxLength);
}

public ArrayList<float[]> getEntityBounds() {
    ArrayList<float[]> entityBounds = new ArrayList<float[]>();

    entityBounds.add(
Geometry.createRectangle((float)(bodyCorners[0][0]+position_WC_x),(float)(bodyCorners[0][1]-position_WC_y),
        (float)(bodyCorners[1][0]+position_WC_x),(float)(bodyCorners[1][1]-position_WC_y),(float)(bodyCorners[2][0]+position_WC_x),(float)(bodyCorners[2][1]-position_WC_y),
        (float)(bodyCorners[3][0]+position_WC_x),(float)(bodyCorners[3][1]-position_WC_y));
}

```

```

if(detectWheels) {
    for(int j = 0; j < 4; j++) {
        float[][] calc = new float[4][2];
        calc[0][0] = -wheeldepth/2;
        calc[0][1] = wheeldiameter/2;

        calc[1][0] = wheeldepth/2;
        calc[1][1] = wheeldiameter/2;

        calc[2][0] = wheeldepth/2;
        calc[2][1] = -wheeldiameter/2;

        calc[3][0] = -wheeldepth/2;
        calc[3][1] = -wheeldiameter/2;

        for(int i = 0; i < 4; i++) {
            if(j<2) {
                calc[i][0] =
(float)(cos(angle+steerangle)*calc[i][0] - sin(angle+steerangle)*calc[i][1]);
                calc[i][1] =
(float)(sin(angle+steerangle)*calc[i][0] + cos(angle+steerangle)*calc[i][1]);
            } else {
                calc[i][0] = (float)(cos(angle)*calc[i][0] -
sin(angle)*calc[i][1]);
                calc[i][1] = (float)(sin(angle)*calc[i][0] +
cos(angle)*calc[i][1]);
            }
            calc[i][0] += wheelCenters[j][0];
            calc[i][1] += wheelCenters[j][1];
        }
        entityBounds.add(
Geometry.createRectangle((float)(calc[0][0]+position_WC_x),(float)(calc[0][1]-
position_WC_y),
(floating)(calc[1][0]+position_WC_x),(floating)(calc[1][1]-position_WC_y),
(floating)(calc[2][0]+position_WC_x),(floating)(calc[2][1]-position_WC_y),
(floating)(calc[3][0]+position_WC_x),(floating)(calc[3][1]-position_WC_y)) );
    }
}
return entityBounds;
}

```

```

// public void setObstacles( ArrayList<float[]> obstacleSet ) {
    // chkList.clear();
    // for(int i=0; i < obstacleSet.size(); i++) {
        //
        if(Geometry.length(obstacle.get(i).getCoords(),getAbsCoords())<radarRange)
            // chkList.add(obstacleSet.get(i));
        //
    // }
    // chkList.addAll(obstacleSet);

    //
}

public int getRRTIndicator() {
    return embeddedHardware.getRRT_Enable();
}

/// \todo Draw in vehicle velocity vector too (relative to car frame and relative to
global frame)
public void updateSteerFromRRT() {
    if(embeddedHardware.processingData) { // && previous RRT/threat was
enabled remain
        try{
            processingThread.join();
        } catch(Exception e) {}
    }
    else {
        // System.out.println("Processed in time");
    }
    // get value from read in buffer of embeddedHardwareObject
    if(embeddedHardware.getRRT_Enable() > 0) {
        //System.out.println();
        // System.out.println("RRT_ENABLE:
"+embeddedHardware.getRRT_Steer()); // \todo Enable RRT LED on dashboard
        setSteerangle(
(float)(embeddedHardware.getRRT_Steer()*steerGearReduct*(PI/180.0f)) );
        //System.out.println();
    }
}

public void updateLiDAR(double elaspedTime) {
    /// \todo Make lidarEnabled determine whether Lidar code is run for each
vehicle not if AI controlled or not

    if(!AIControlled) { // && lidarEnabled) {
        if(lidar.lastScan >= lidar.lidarScanRate) {
            lidar.scan();
            processData();
}

```

```

        lidar.lastScan = 0;
    }
    lidar.lastScan += elaspedTime;
}
}

private void updateSpeedController() {
    // If PID speedcontroller is enabled calculate output
    if(speedController!=null) {
        if(speedController.getEnable()) {
            // Calculate throttle output (%) based on error and PID loop
            float tmpThrottle =
speedController.calculateOutput(speed);
            // Check for saturation
            if(tmpThrottle > 100)
                tmpThrottle = 100;
            else if(tmpThrottle < 0)
                tmpThrottle = 0;
            // Set throttle to controller throttle output
            throttle = tmpThrottle;
        } else {
            if(speedController.getLastEnable()==1)
                throttle = 0;
        }
    }
}

private void processData() {
    // Set x-coords of point-cloud buffer from simulated scan
    embeddedHardware.setBufferXFloat(lidar.getPointCloudYPts()); // SAE
coordinate system to senior project coordinate system
    // Set y-coords of point-cloud buffer from simulated scan
    embeddedHardware.setBufferYFloat(lidar.getPointCloudXPts()); // SAE
coordinate system to senior project coordinate system
    // Convert steer angle in rad to PWM value for steer servo motor control

    embeddedHardware.setSteerAngle((byte)(steerangle*(180/PI)*7.5f+127.67f));
    // Convert throttle percentage to PWM value for eletronic speed controller
(ESC) for DC brushless motor control
    embeddedHardware.setThrottle((byte)((throttle-brake)*1.27f+128));
    // Set vehicle x-position buffer
    embeddedHardware.setBufferPosX((float)position_WC_x);
    // Set vehicle y-position buffer
    embeddedHardware.setBufferPosY((float)position_WC_y);
    // Set vehicle angle buffer
    embeddedHardware.setBufferAng((float)angle);
}

```

```

        // If not already processing data set flag true and start a new thread that
        writes buffers to serial stream and waits
        // for embedded target to respond (see serial event function)
        if(!embeddedHardware.processingData) {
            embeddedHardware.processingData = true;
            processingThread = (new Thread(embeddedHardware));
            processingThread.start();
        }
    }

    private void calculateManueverAngle() {
        Line2D.Float carLine = new
        Line2D.Float((float)getX(),(float)getY(),(float)(getX())+cos(angle-
PI/2),(float)(getY())+sin(angle-PI/2));
        int centerLineindex = 0;
        float smallestDistance = 0f;
        if(path.size()>0) {
            for(int i=0; i<path.size(); i++) {
                Line2D.Float centerLine = path.get(i);
                float tempLength = (float)centerLine.ptSegDist(getX(),getY());
                if((tempLength<smallestDistance)|| (i==0)) {
                    smallestDistance = tempLength;
                    centerLineindex = i;
                }
            }
        }
        Line2D.Float closetPathline = path.get(centerLineindex);

        float angBet =(float)
        (signum(angle)*Geometry.angleBetween2Lines(closetPathline, carLine));
        float alpha;
        if(smallestDistance < lfw)
            alpha = -angBet-(float)(PI/2-acos(signum(getY()-
closetPathline.getP1().getY())*smallestDistance/lfw));
        else
            alpha = -angBet;
        steerangle = (float)atan2(2*wheelbase*sin(alpha),lfw);
    }
}

private void updateRealTimePlotterData() {
    for(int i = 0; i <= graphNum-2; i++) {
        Vxpts[i] = Vxpts[i+1];
        Vypts[i] = Vypts[i+1];
        yawRatepts[i] = yawRatepts[i+1];
        SteerAngle[i] = SteerAngle[i+1];
    }
    Vxpts[graphNum-1] = (float)getvLong();
}

```

```

        if( abs(Vxpts[graphNum-1]) > VxPtsMax )
            VxPtsMax = abs(Vxpts[graphNum-1]);
        Vypts[graphNum-1] = (float)getvLat();
        if( abs(Vypts[graphNum-1]) > VyPtsMax )
            VyPtsMax = abs(Vypts[graphNum-1]);
        yawRatepts[graphNum-1] = (float)getYawRate();
        if( abs(yawRatepts[graphNum-1]) > yawRateptsMax )
            yawRateptsMax = abs(yawRatepts[graphNum-1]);
        SteerAngle[graphNum-1] = getSteerAngle();
    }

public synchronized void update(double elaspedTime) {
    // If PID speedcontroller is enabled calculate output
    updateSpeedController();
    // If AI Maneuver is enabled calculate maneuver to follow path
    if( (path != null) && enableManuever ) {
        if(!AIControlled) {
            if(embeddedHardware.getRRT_Enable()!=1)
                calculateManueverAngle();
        } else
            calculateManueverAngle();
    }
    // If realtime plotter is enabled - update plotter data
    updateRealTimePlotterData();

    // UPDATE VEHICLE PHYSICS
    // Low-speed model (Ackermann) flag
    boolean ackermann = false;
    double sn = sin(angle);
    double cs = cos(angle);
    // Global frame to car frame - calculate velocity and speed
    velocity_x = (float)(cs*velocity_WC_y + sn*velocity_WC_x);
    velocity_y = (float)(-sn*velocity_WC_y + cs*velocity_WC_x);
    speed = (float)sqrt( pow(velocity_x,2) + pow(velocity_y,2) );
    // Average yaw speed of COG, front, and rear
    yawspeed = wheelbase*0.5f*angularvelocity;
    yawspeed_f = F2CG*angularvelocity;
    yawspeed_r = R2CG*angularvelocity;
    // Initialize magic formula lateral forces
    flatf_x = 0;
    flatr_x = 0;
    // Initialize ackermann forces
    float angular_speed_ackermann = 0;
    /// \todo Implement braking model - (however not critical for this project)
    // brake = 0;
}

```

```

// MOTOR CALCULATIONS
motorModel.setMotorState(abs(velocity_x), throttle-brake);
// Rolling Resistance and Drag
if(speed > 0)
    fr = 0.01f*(1.0f+(speed*2.23694f)/100.0f); // coefficient - speed
influence
else
    fr = 0;
resistance_x = -( DRAG*velocity_x*abs(velocity_x) +
fr*weight*signum(velocity_x) );
resistance_y = -( DRAG*velocity_y*abs(velocity_y) +
fr*weight*signum(velocity_y) );

// PEAK TRACTIVE FORCE @ DRIVE WHEEL
// if(RWD) {
// ftraction_y = 0;
// else if(FWD)
// ftraction_x = 0;
ftraction_y = 0;
ftraction_x = powerTrainEff*motorModel.getMotorTractiveForce();

if(abs(velocity_x) <= LOW_SPEED_THRES) {
    // low-speed use ackermann model - negligible slip angles and
lateral forces, geometric turning
    angular_speed_ackermann =
(float)((speed/wheelbase)*tan(steerangle));
    ackermann = true;
}
else {
    // Kinetic Steering - Lateral forces estimated with 'Magic' Formula,
tire deformation, and slip & side-slip.
    rot_angle = (float)atan2( yawspeed, velocity_x );
    rot_angle_front = (float)atan2( yawspeed_f, velocity_x );
    rot_angle_rear = (float)atan2( yawspeed_r, velocity_x );
    // Global Slip Angle
    sideslip = (float)atan2( velocity_y, velocity_x );
    // Front Slip Angle
    slipanglefront = sideslip + rot_angle_front - steerangle;
    // Rear Slip Angle
    slipanglerear = sideslip - rot_angle_rear;

    if(tireModel.model == TireModel.MODEL.LINEAR) {
        flatf_y = 2*tireModel.CA_F*slipanglefront;
        flatr_y = 2*tireModel.CA_R*slipanglerear;
    } else {
        flatf_y = 2*tireModel.getLateralForce(slipanglefront);
    }
}
}

```

```

        flatr_y = 2*tireModel.getLateralForce(slipanglerear);
    }
    // Tire lateral force dependence on tire's normal force (i.e. D =
mew * Fz)
    flatf_y *= (Nf/weight);
    flatr_y *= (Nr/weight);
}

// FRICTION CIRCLE
double ftraction_limit = tireModel.getDynamicFrictionCoeff(speed)*Nr;
if(abs(ftraction_x) > ftraction_limit) {
    // System.out.println("REAR SLIDE");
    ftraction_x = signum(ftraction_x)*ftraction_limit;
    rear_slide = 1;
} else
    rear_slide = 0;
double flat_limit_driveWheel = sqrt(pow(fraction_limit,2)-
pow(fraction_x,2));
if(abs(flatf_y) > ftraction_limit) {
    flatf_y = signum(flatf_y)*ftraction_limit;
    // System.out.println("FRONT SLIDE");
    front_slide = 1;
} else
    front_slide = 0;
if(abs(flatr_y) > flat_limit_driveWheel) {
    flatr_y = signum(flatr_y)*flat_limit_driveWheel;
    // System.out.println("REAR SLIDE - LAT");
    rear_slide = 1;
} else
    rear_slide = 0;

if(!ackermann) {
    force_x = ftraction_x + resistance_x + (float)sin(steerangle) *
flatf_x + flatr_x;
    force_y = ftraction_y + resistance_y + (float)cos(steerangle) *
flatf_y + flatr_y;
} else {
    float pho = (float)(wheelbase/tan(steerangle));
    float lateralF = (float)(mass*pow(speed,2))/pho;
    force_x = ftraction_x + resistance_x;
    force_y = ftraction_y + resistance_y + lateralF;
    flatf_x = 0;
    flatr_x = 0;
    flatf_y = 0;
    flatr_y = 0;
}
}

```

```

torque = F2CG*flatf_y - R2CG*flatr_y;

acceleration_x = force_x/mass;
acceleration_y = force_y/mass;

// Evaluate Tire Normal Forces for Next Update
Nf = (weight*R2CG-mass*acceleration_x*h)/wheelbase;
Nr = weight - Nf;

if(!ackermann)
    angular_acceleration = torque/inertia;
// Transform from car coords to world/global coords with rotation matrix
acceleration_WC_x = (float)(cs*acceleration_y+sn*acceleration_x);
acceleration_WC_y = (float)(-sn*acceleration_y+cs*acceleration_x);

if(updateSpeedFromEncoders&&(!AIControlled)) {
    // velocity_x & velocity_y should equal a component of the
measured velocity * slip angle(simulated)
    float speedMeas = embeddedHardware.getMeasuredSpeed();

    float sim_velocity_WC_x = (float)(velocity_WC_x +
elapsedTime*acceleration_WC_x);
    float sim_velocity_WC_y = (float)(velocity_WC_y +
elapsedTime*acceleration_WC_y);
    float sim_position_WC_x = (float)(position_WC_x +
elapsedTime*velocity_WC_x);
    float sim_position_WC_y = (float)(position_WC_y +
elapsedTime*velocity_WC_y);
    float sim_angularvelocity = (float)(angularvelocity +
elapsedTime*angular_acceleration);
    float sim_angle = (float)(angle + elapsedTime*angularvelocity);
    // Rotation matrix - repeated calculation once here for efficiency
    float sim_sn = (float)sin(sim_angle);
    float sim_cs = (float)cos(sim_angle);
    // Global frame to car frame - calculate velocity and speed
    float sim_velocity_x = (float)(sim_cs*sim_velocity_WC_y +
sim_sn*sim_velocity_WC_x);
    float sim_velocity_y = (float)(-sim_sn*sim_velocity_WC_y +
sim_cs*sim_velocity_WC_x);
    float sim_side_slip;

    if(abs(sim_velocity_x) <= LOW_SPEED_THRES) {
        sim_side_slip = 0;
    }
    else {
        sim_side_slip = (float)atan2( sim_velocity_y,
sim_velocity_x );
    }
}

```

```

        // estimate slipangle by propagating one time step and evaluating
        side slip angle with propagated, simulated velocities
        velocity_x = (float)(cos(sim_side_slip)*speedMeas);
        velocity_y = (float)(sin(sim_side_slip)*speedMeas);
        // transform vehicle reference frame to global reference frame
        velocity_WC_x = (float)(cs*velocity_y + sn*velocity_x);
        velocity_WC_y = (float)(-sn*velocity_y + cs*velocity_x);
    }
    else {
        // Euler integration scheme - velocity
        velocity_WC_x += elaspedTime*acceleration_WC_x;
        velocity_WC_y += elaspedTime*acceleration_WC_y;
    }
    // Euler integration scheme - position
    position_WC_x += elaspedTime*velocity_WC_x;
    position_WC_y += elaspedTime*velocity_WC_y;

    if(!ackermann) {
        angularvelocity += elaspedTime*angular_acceleration;
        angle += elaspedTime*angularvelocity;
    } else {
        angularvelocity = angular_speed_ackermann;
        angle += angular_speed_ackermann*elaspedTime;
    }

    add_to_trail( (float)position_WC_x, (float)position_WC_y, (float)angle );
}

public synchronized void disconnect() {
    embeddedHardware.disconnect();
}
public void closeRecordStream() {
    try {
        recordStream.close();
    } catch(Exception e) { SimData.logger.warning("Failed to close car record
stream(" + e.getStackTrace() + ")"); }
}
private void draw_trail(Graphics2D g)
{
    float x,y;
    for(int i = 0; i < num_trail; i++)
    {
        x = (float)((trail[i][0]-
position_WC_x)*SimData.scale+screen_pos_x);
        y = (float)(-(trail[i][1]-
position_WC_y)*SimData.scale+screen_pos_y);

```

```

        g.drawOval((int)x,(int)y,2,2);
    }
}

private void drawCarTexture(Graphics2D g2d) {
    if(image.size() > 0) {
        g2d.translate( screen_pos_x, screen_pos_y );
        g2d.rotate( angle );//PI/2 );
        g2d.drawImage( getImage(), (int)(-(width/2)*SimData.scale*2),
(int)(-(length/2)*SimData.scale*1.5), (int)(width*SimData.scale*2),
(int)(length*SimData.scale*1.5), null );
        g2d.rotate( -angle );//+PI/2 );
        g2d.translate( -screen_pos_x, -screen_pos_y );
    }
}

private void drawSlideTexture(Graphics2D g2d) {
    if(image.size() > 2) {
        //if(front_slide==1) {
            g2d.translate(screen_pos_x, screen_pos_y);
            g2d.rotate( angle-PI/2 );

            if(front_slide == 1) {
                g2d.translate( R2CG*SimData.scale,
(width/2)*SimData.scale );
                g2d.rotate( steerangle );
                for(int i = 0;i<fslideCount;i++)
                    g2d.drawImage( getImage(2), (int)(-
(i+1.5)*wheeldiameter*SimData.scale), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null);
                g2d.rotate( -steerangle );
                g2d.translate( -R2CG*SimData.scale, -
(width/2)*SimData.scale );

                g2d.translate( R2CG*SimData.scale, -
(width/2)*SimData.scale );
                g2d.rotate( steerangle );
                for(int i = 0;i<fslideCount;i++)
                    g2d.drawImage( getImage(2), (int)(-
(i+1.5)*wheeldiameter*SimData.scale), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null);
                g2d.rotate( -steerangle );
                g2d.translate( -R2CG*SimData.scale,
(width/2)*SimData.scale );
                fslideCount++;
            } else {
                if(fslideCount>0)
                    fslideCount--;
            }
        }
    }
}

```

```

        }
        if(rear_slide == 1) {
            g2d.translate( -F2CG*SimData.scale, -
(width/2)*SimData.scale );
            for(int i = 0;i<rslideCount;i++)
                g2d.drawImage( getImage(2), (int)(-
(i+1.5)*wheeldiameter*SimData.scale), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null);
            g2d.translate( F2CG*SimData.scale,
(width/2)*SimData.scale );

            g2d.translate( -F2CG*SimData.scale,
(width/2)*SimData.scale );
            for(int i = 0;i<rslideCount;i++)
                g2d.drawImage( getImage(2), (int)(-
(i+1.5)*wheeldiameter*SimData.scale), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null);
            g2d.translate( F2CG*SimData.scale, -
(width/2)*SimData.scale );
            rslideCount++;
        } else {
            if(rslideCount > 0)
                rslideCount--;
        }

        g2d.rotate( -angle+PI/2 );
        g2d.translate(-screen_pos_x, -screen_pos_y);
        // slideCount++;
    //}
}
}

private void drawWheelTexture(Graphics2D g2d) {
    if(image.size() > 1) {
        for(int i = 0; i < 4; i++) {
            g2d.translate( screen_pos_x, screen_pos_y );
            g2d.rotate( angle-PI/2 );
            switch(i) {
                case 0:
                    g2d.translate( -F2CG*SimData.scale, -
(width/2)*SimData.scale );
                    g2d.drawImage( getImage(1), (int)(-
((wheeldiameter*SimData.scale)/2)), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null );
                    g2d.translate( F2CG*SimData.scale,
(width/2)*SimData.scale );
                    break;
            }
        }
    }
}

```

```

        case 1:
            g2d.translate( -F2CG*SimData.scale,
(width/2)*SimData.scale );
            g2d.drawImage( getImage(1), (int)(-
((wheeldiameter*SimData.scale)/2)), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null );
            g2d.translate( F2CG*SimData.scale, -
(width/2)*SimData.scale );
            break;
        case 2:
            g2d.translate( R2CG*SimData.scale,
(width/2)*SimData.scale );
            g2d.rotate( steerangle );
            g2d.drawImage( getImage(1), (int)(-
((wheeldiameter*SimData.scale)/2)), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null );
            g2d.rotate( -steerangle );
            g2d.translate( -R2CG*SimData.scale, -
(width/2)*SimData.scale );
            break;
        case 3:
            g2d.translate( R2CG*SimData.scale, -
(width/2)*SimData.scale );
            g2d.rotate( steerangle );
            g2d.drawImage( getImage(1), (int)(-
((wheeldiameter*SimData.scale)/2)), (int)(-((wheeldepth*SimData.scale)/2)),
(int)(wheeldiameter*SimData.scale), (int)(wheeldepth*SimData.scale), null );
            g2d.rotate( -steerangle );
            g2d.translate( -R2CG*SimData.scale,
(width/2)*SimData.scale );
            break;
    }
}
}

private void add_to_trail( float x, float y, float angle)
{
    if( num_trail < TRAIL_SIZE-1 )
    {
        trail[num_trail][0] = x;
        trail[num_trail][1] = y;
        trail[num_trail][2] = angle;
        num_trail++;
    }
}

```

```

    }
else
{
    for(int i=0; i < TRAIL_SIZE-1; i++) {
        trail[i][0] = trail[i+1][0];
        trail[i][1] = trail[i+1][1];
        trail[i][2] = trail[i+1][2];
    }
    trail[num_trail][0] = x;
    trail[num_trail][1] = y;
    trail[num_trail][2] = angle;
}
}

private void draw_rect( Graphics2D g, float angle, float w, float l, float x, float y,
int crossed) {
    // draw_rect( g, angle+(nr<2 ? steerangle : 0), wheeldepth*SimData.scale,
    wheeldiameter*SimData.scale, wheels[0][0], wheels[0][1], crossed );
    float[][] c = new float[4][2];
    float[][] c2 = new float[4][2];
    double sn = sin(angle);
    double cs = cos(angle);

    c[0][0] = -w/2;
    c[0][1] = l/2;

    c[1][0] = w/2;
    c[1][1] = l/2;

    c[2][0] = w/2;
    c[2][1] = -l/2;

    c[3][0] = -w/2;
    c[3][1] = -l/2;

    for(int i = 0; i <= 3; i++) {
        c2[i][0] = (float)(cs*c[i][0] - sn*c[i][1]);
        c2[i][1] = (float)(sn*c[i][0] + cs*c[i][1]);
        c[i][0] = c2[i][0];
        c[i][1] = c2[i][1];
    }

    for(int i = 0; i <= 3; i++) {
        c[i][0] += x;
        c[i][1] += y;
    }
}

```

```

        g.draw( new Line2D.Double(c[0][0], c[0][1], c[1][0], c[1][1]) );
        g.draw( new Line2D.Double(c[1][0], c[1][1], c[2][0], c[2][1]) );
        g.draw( new Line2D.Double(c[2][0], c[2][1], c[3][0], c[3][1]) );
        g.draw( new Line2D.Double(c[3][0], c[3][1], c[0][0], c[0][1]) );

        if(crossed==1) {
            g.draw( new Line2D.Double(c[0][0], c[0][1], c[2][0], c[2][1]) );
            g.draw( new Line2D.Double(c[1][0], c[1][1], c[3][0], c[3][1]) );
        }
    }

    public synchronized void writeCarToFile(String filepath) {
        try {
            // Serialize data object to a file
            ObjectOutputStream CarOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
            CarOut.writeObject(this);
            CarOut.close();

            //Serialize data object to a byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            CarOut = new ObjectOutputStream(bos);
            CarOut.writeObject(this);
            CarOut.close();
            byte[] buf = bos.toByteArray();
            SimData.logger.info("Save car successfully");
        } catch (IOException e) {
            SimData.logger.warning("Failed to write car object to file (" +
e.toString() + ")");
        }
    }

    public void setToolBarHeight(int toolBarHeight ) {
        this.toolBarHeight = toolBarHeight;
    }

    /**
     * This functions sets the throttle (in %)
     * \param throttle Percent throttle to actuate
     */
    public synchronized void setThrottle(float throttle) {
        this.throttle = throttle;
    }

    /**
     * This functions returns the throttle (in %)
     * \return throttle Percent throttle to actuate
     */
    public synchronized float getThrottle() {
        return throttle;
    }
}

```

```

/** This functions sets the steerangle (in rads)
 * \param steerangle Sets the current car steerangle
 */
public synchronized void setSteerangle(float steerangle) {
    float reducedSteer = steerangle/steerGearReduct;
    prevsteerangle = steerangle;
    if(abs(reducedSteer) > STEER_LIMIT*(PI/180))
        this.steerangle =
(float)(signum(reducedSteer)*STEER_LIMIT*(PI/180));
    else
        this.steerangle = reducedSteer;
}
/** This functions sets the brakes (in %)
 * \param brakes Percent brakes to actuate
 */
public void setBrakes(float brakes) {
    brake = brakes;
}

public boolean isFrontSliding() {
    if(front_slide == 1)
        return true;
    else
        return false;
}
public boolean isRearSliding() {
    if(rear_slide == 1)
        return true;
    else
        return false;
}
/// \todo Implement different camera perspectives (i.e. follow car, static, static
except near edges...)
/** This functions returns the brakes (in %)
 * \return brakes Percent brakes to actuate
 */
public float getBrakes() {
    return brake;
}
/** This functions sets the car's scale
 * \param scale New car scale
 */
public void setScale(float newscaleoverOldscale) {
    // adjust positions
    position_WC_x *= newscaleoverOldscale;
    position_WC_y *= newscaleoverOldscale;

```

```

    }
    /** This functions returns the car's x position on the screen
     * \return Car's x position
     */
    public double getScreenXPos() {
        return screen_pos_x;
    }
    /** This functions returns the car's y position on the screen
     * \return Car's y position
     */
    public double getScreenYPos() {
        return screen_pos_y;
    }
    /** This functions returns the car's absolute coords (in pixels) from the origin
     * (i.e. the center of the screen)
     * \return Car's absolute coordinates
     */
    public float[] getAbsCoords() {
        return new float[] { (float)(position_WC_x*SimData.scale +
(screenSize.getWidth()/2)) , (float)(-position_WC_y * SimData.scale +
(screenSize.getHeight()/2)) };
    }
    /** This functions returns the car's speed in mph
     * \return Car's speed in mph
     */
    public float getSpeed() {
        return speed*2.23694f; // return speed in mph
    }
    /** This functions returns the car's x-position in world coordinates with units
     * of meters
     * \return Car's x-position in world coordinates in meters
     */
    public double getX() {
        return position_WC_x;
    }
    public synchronized void setX(float position_WC_x) {
        this.position_WC_x = position_WC_x;
    }
    /** This functions returns the car's y-position in world coordinates with units
     * of meters
     * \return Car's y-position in world coordinates in meters
     */
    public double getY() {
        return position_WC_y;
    }
    public synchronized void setY(float position_WC_y) {

```

```

        this.position_WC_y = position_WC_y;
    }

    private synchronized double getAngle() {
        return angle;
    }

    private double getvLong() {
        return velocity_x;
    }

    private double getvLat() {
        return velocity_y;
    }

    private double getYawRate() {
        return angularvelocity;
    }

    private synchronized float getSteerAngle() {
        return steerangle;
    }

    private void draw_wheel( Graphics2D g, int nr, float x, float y, int crossed) {
        draw_rect( g, (float)(angle+(nr<2 ? steerangle : 0)),
wheeldepth*SimData.scale, wheeldiameter*SimData.scale, x, y, crossed );
    }

    public synchronized void keyPressed(int key) {
        if(key == KeyEvent.VK_LEFT) { if(steerangle > -
STEER_LIMIT*(PI/180.0f)) { if(steerangle<-STEER_LIMIT*(PI/180.0f)) {
            steerangle = (float)(-STEER_LIMIT*(PI/180.0f)); } else {
steerangle -= PI/32.0f; } } }
        if(key == KeyEvent.VK_RIGHT) { if(steerangle <
STEER_LIMIT*(PI/180.0f)) { if(steerangle> STEER_LIMIT*(PI/180.0f)) {
            steerangle = (float)(STEER_LIMIT*(PI/180.0f)); } else {
steerangle += PI/32.0f; } } }
        if(key == KeyEvent.VK_UP) {
            throttle = 100;
            brake = 0;
        }
        if(key == KeyEvent.VK_DOWN) {
            brake = 100;
            throttle = 0;
        }
    }
}

```

```

public synchronized void keyReleased(int key) {
    //if(key == KeyEvent.VK_LEFT) { steerAngle = 0; }
    //if(key == KeyEvent.VK_RIGHT){ steerAngle = 0;      }
    if(key == KeyEvent.VK_UP) {   throttle =      0;  }
    if(key == KeyEvent.VK_DOWN) { brake =        0;  }
}

public void drawRealTimePlotter(Graphics2D g2d, int plotter_x, int plotter_y) {
    Composite c = g2d.getComposite();

    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
0.3f));
    g2d.setColor(Color.black);
    g2d.fillRect(plotter_x , plotter_y, realTimePlotterWidth,
realTimePlotterHeight);
    g2d.setComposite(c);
    g2d.translate( plotter_x, realTimePlotterHeight/2+plotter_y );
    if(!interpPlot) {
        for(int i = 0; i < graphNum; i++) {

            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),0,1,Color.lightGray);
            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),
(Vxpts[i]/VxPtsMax)*(realTimePlotterHeight/2),1,Color.yellow);
            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),
(Vypts[i]/VyPtsMax)*(realTimePlotterHeight/2),1,Color.white);

            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),(float)(-
(pow(pow(Vxpts[i],2)+pow(Vypts[i],2),0.5)/pow(pow(VyPtsMax,2)+pow(VxPtsMax,2),
0.5))*(realTimePlotterHeight/2)),1,Color.orange);

            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),(float)(-
((SteerAngle[i]*(180/PI))/STEER_LIMIT)*(realTimePlotterHeight/2)),1,Color.green);
            drawPixel(g2d,i*(realTimePlotterWidth/graphNum),
(yawRatepts[i]/yawRateptsMax)*(realTimePlotterHeight/2),1,Color.pink);
        }
    } else {
        for(int i = 0; i < graphNum-1; i++) {
            g2d.setColor(Color.lightGray);
            // \todo replace with g.drawLine( new Line2D.Double(
                g2d.drawLine(i*(realTimePlotterWidth/graphNum),0,(i+1)*(realTimePlotterWidth/graphNum),0);
            g2d.setColor(Color.yellow);
            // \todo replace with g.drawLine( new Line2D.Double(

```

```

        g2d.drawLine(i*(realTimePlotterWidth/graphNum),(int)(-
(Vxpts[i]/VxPtsMax)*(realTimePlotterHeight/2)),(i+1)*(realTimePlotterWidth/graphNu-
m),(int)(-(Vxpts[i+1]/VxPtsMax)*(realTimePlotterHeight/2)));
        g2d.setColor(Color.white);
        /// \todo replace with g.draw( new Line2D.Double(
        g2d.drawLine(i*(realTimePlotterWidth/graphNum),(int)(-
(Vypts[i]/VyPtsMax)*(realTimePlotterHeight/2)),(i+1)*(realTimePlotterWidth/graphNu-
m),(int)(-(Vypts[i+1]/VyPtsMax)*(realTimePlotterHeight/2)));
        g2d.setColor(Color.orange);
        /// \todo replace with g.draw( new Line2D.Double(
        g2d.drawLine(i*(realTimePlotterWidth/graphNum),(int)(-
(sqrt(pow(Vxpts[i],2)+pow(Vypts[i],2))/sqrt(pow(VyPtsMax,2)+pow(VxPtsMax,2)))*(re-
alTimePlotterHeight/2)),(i+1)*(realTimePlotterWidth/graphNum),
(int)(-
(sqrt(pow(Vxpts[i+1],2)+pow(Vypts[i+1],2))/sqrt(pow(VyPtsMax,2)+pow(VxPtsMax,2)))
)*(realTimePlotterHeight/2));
        g2d.setColor(Color.green);
        /// \todo replace with g.draw( new Line2D.Double(
        g2d.drawLine(i*(realTimePlotterWidth/graphNum),(int)( -
((SteerAngle[i]*(180/PI))/STEER_LIMIT)*(realTimePlotterHeight/2)),(i+1)*(realTimeP-
lotterWidth/graphNum),(int)(-
((SteerAngle[i+1]*(180/PI))/STEER_LIMIT)*(realTimePlotterHeight/2)));
        g2d.setColor(Color.pink);
        /// \todo replace with g.draw( new Line2D.Double(
        g2d.drawLine(i*(realTimePlotterWidth/graphNum),(int)( -
(yawRatepts[i]/yawRateptsMax)*(realTimePlotterHeight/2)
),(i+1)*(realTimePlotterWidth/graphNum),(int)(-
(yawRatepts[i+1]/yawRateptsMax)*(realTimePlotterHeight/2)));
        }
    }

    g2d.setColor(Color.yellow);
    g2d.drawString(velocity_x_LEGEND, 0, realTimePlotterHeight/2);
    g2d.setColor(Color.white);
    g2d.drawString(velocity_y_LEGEND,
g2d.getFontMetrics().stringWidth(velocity_x_LEGEND)+5, realTimePlotterHeight/2);
    g2d.setColor(Color.orange);
    g2d.drawString(speed_LEGEND,
g2d.getFontMetrics().stringWidth(velocity_x_LEGEND+velocity_y_LEGEND)+10,
realTimePlotterHeight/2);
    g2d.setColor(Color.green);
    g2d.drawString("Steer",
g2d.getFontMetrics().stringWidth(velocity_x_LEGEND+velocity_y_LEGEND+speed_L-
EGEND)+15, realTimePlotterHeight/2);
    g2d.setColor(Color.pink);

```

```

        g2d.drawString("Yaw Rate",
g2d.getFontMetrics().stringWidth(velocity_x_LEGEND+velocity_y_LEGEND+speed_L
EGEND+"Steer")+20, realTimePlotterHeight/2);

        g2d.translate( -plotter_x, -realTimePlotterHeight/2-plotter_y );
    }

    private static void drawPixel(Graphics2D g, float x, float y, float size, Paint color)
{
    Shape circle = new Ellipse2D.Float(x, y, size, size);
    g.setPaint(color);
    g.draw(circle);
    g.setPaint(color);
    g.fill(circle);
}

<*/
*     \class PIDcontroller
*     \brief The PIDcontroller class handles the controller output and parameters
*/
public class PIDcontroller {
    private float SP;
    private float errorSum;
    private float P;
    private float I;
    private float D;
    private float prevInput;
    private boolean enableOutput = false;
    private int lastEnable = 0;
    private boolean limitIfSliding = true;
    private float lastOutputnoSlide;

    public PIDcontroller() {

    }
    public PIDcontroller(float P) {
        this.P = P;
    }
    public PIDcontroller(float P, float I) {
        this.P = P;
        this.I = I;
    }
    public PIDcontroller(float P, float I, float D) {
        this.P = P;
        this.I = I;
        this.D = D;
    }
}

```

```

public PIDcontroller(float P, float I, float D, float SP) {
    this.P = P;
    this.I = I;
    this.D = D;
    this.SP = SP;
}
public PIDcontroller(float P, float I, float D, float SP, boolean
enableOutput) {
    this.P = P;
    this.I = I;
    this.D = D;
    this.SP = SP;
    this.enableOutput = enableOutput;
}

public void setSP(float SP) {
    this.SP = SP;
}
public void setP(float P) {
    this.P = P;
}
public void setI(float I) {
    this.I = I;
}
public void setD(float D) {
    this.D = D;
}
public void setEnable(boolean enableOutput) {
    if(this.enableOutput && this.lastEnable!=-1)
        this.lastEnable = 1;
    else if(this.lastEnable!=-1)
        this.lastEnable = 0;
    this.enableOutput = enableOutput;
}
public boolean getEnable() {
    return enableOutput;
}
public void setLastEnable(int lastEnable) {
    this.lastEnable = lastEnable;
}
public int getLastEnable() {
    return lastEnable;
}
public float calculateOutput(float sensorInput) {
    float error = SP - sensorInput;
    float ROC = sensorInput - prevInput;
}

```

```

        errorSum += error;

        float output = P*error + I*errorSum + D*ROC;

        prevInput = sensorInput;

        return output;
    }
}

/**
 * \class LiDAR
 * \brief Simulated LiDAR sensor with adjustable range, resolution,
 *        and scan frequency in order to identify threatening obstacles
 *        within the simulated environment - provides distance and relative
velocity
 *
 *        of frontal obstacle (no mechanical/phased array scanning feature)
 */
public class Radar {
    public Radar() {

    }

}

/**
 * \class LiDAR
 * \brief Simulated LiDAR sensor with adjustable range, resolution,
 *        and scan frequency in order to identify threatening obstacles
 *        within the simulated environment
 */
public class LiDAR implements Recordable {
    private ArrayList<float[]> chkList = new ArrayList<float[]>();
    private ArrayList<Float> x = new ArrayList<Float>();
    //> Current Point-Cloud (x-points)
    private ArrayList<Float> y = new ArrayList<Float>();
    //> Current Point-Cloud (y-points)
    private ArrayList<Float> x_transformed = new ArrayList<Float>();
    //> Current Point-Cloud (x-points)
    private ArrayList<Float> y_transformed = new ArrayList<Float>();
    //> Current Point-Cloud (y-points)
    private Float[] prevx;
    //> Previous Point-Cloud (x-points)
    private Float[] prevy;
    //> Previous Point-Cloud (y-points)
protected float radarRange;
protected float radarAngle;

```

```

protected float lastScan;
protected float lidarScanRate = 0.05f;
protected float radarAngularRange = 240f;
protected float radarAngleRangeFactor;
protected int numberBeams = 61;
protected float increment = (radarAngularRange/numberBeams);

public LiDAR(float radarRange) {
    this.radarRange = radarRange;
    radarAngleRangeFactor = (360f-radarAngularRange)/2;
}
public LiDAR(float radarRange, float radarAngularRange, float
lidarScanRate, int numberBeams) {
    this.radarRange = radarRange;
    this.radarAngularRange = radarAngularRange;
    this.lidarScanRate = lidarScanRate;
    this.numberBeams = numberBeams;

    increment = (radarAngularRange/numberBeams);
    radarAngleRangeFactor = (360f-radarAngularRange)/2;
}
public ArrayList<Float> getPointCloudXPs() {
    return x_transformed;
}
public ArrayList<Float> getPointCloudYPs() {
    return y_transformed;
}
public void record() {
    try {

        pointCloud.write(Double.toString(SimData.getSimTime()));
        pointCloud.newLine();
        for(int i=0; i < x.size(); i++) {
            pointCloud.write(x_transformed.get(i)+""
"+y_transformed.get(i));
            pointCloud.newLine();
        }
        pointCloud.newLine();
        pointCloud.flush();
    } catch( IOException x) {
        SimData.logger.warning("Failed to write header to car data
record file ("+x+"));
    } catch(Exception otherExceptions) {
        SimData.logger.warning("Failed to write to car data record
file ("+otherExceptions+"));
    }
}

```

```

        }
        public void setObstacles( ArrayList<float[]> obstacleSet ) {
            chkList.clear();
            chkList.addAll(obstacleSet);
        }
        private void drawPointCloud( Graphics2D g2d ) {
            g2d.translate( screen_pos_x, screen_pos_y );
            for(int i = 0; i < x.size()-1; i++) {
                float x_get,y_get,x_get_1,y_get_1;
                float prevx_get,prevy_get,prevx_get_1,prevy_get_1;
                x_get = x.get(i);
                x_get_1 = x.get(i+1);
                y_get = y.get(i);
                y_get_1 = y.get(i+1);
                try {
                    prevx_get = prevx[i];
                    prevx_get_1 = prevx[i+1];
                    prevy_get = prevy[i];
                    prevy_get_1 = prevy[i+1];
                }
                catch(Exception NoVector) {
                    prevx_get = x_get;
                    prevx_get_1 = x_get_1;
                    prevy_get = y_get;
                    prevy_get_1 = y_get_1;
                }
                Color transBlack = new Color( 255, 255, 255, 125 );
                g2d.setColor(transBlack);
                // replace with g.draw( new Line2D.Double(... ?
                g2d.drawLine(
                    (int)(prevx_get*SimData.scale),(int)(prevy_get*SimData.scale),(int)(prevx_get_1*SimData.scale),
                    (int)(prevy_get_1*SimData.scale));
                g2d.setColor(Color.BLACK);
                // replace with g.draw( new Line2D.Double(... ?
                g2d.drawLine(
                    (int)(x_get*SimData.scale),(int)(y_get*SimData.scale),(int)(x_get_1*SimData.scale),(int)
                    (y_get_1*SimData.scale)); //Draw Original Scan 0-360 degrees
                g2d.setColor(Color.YELLOW);
                // replace with g.draw( new Line2D.Double(... ?
                g2d.drawOval( (int)((x_get*SimData.scale-
                    0.5)),(int)((y_get*SimData.scale-0.5)),1,1 );
            }
            int strokeNum=0;
            for(int i = 0; i < x.size(); i++) {
                float x_get,y_get;
                x_get = x.get(i);

```

```

y_get = y.get(i);
if((i==0)||(i==(x.size()-1))) {
    g2d.setColor(Color.BLACK);
}
else {
    if(strokeNum==0) {
        strokeNum = 1;
        g2d.setColor(Color.CYAN);
    }
    else {
        strokeNum = 0;
        g2d.setColor(new Color( 192,192,192 ));
    }
}

/// \todo replace with g.drawLine( new Line2D.Double(
/// \todo ( placeLidar_x, placeLidar_y, placeLidar_angle )
to allow various placement on vehicle

//g2d.drawLine(
0,0,(int)(x_get*SimData.scale),(int)(y_get*SimData.scale) );
g2d.drawLine( (int)(placeLidar_x*cos(angle)-
placeLidar_y*sin(angle)*SimData.scale),
(int)(placeLidar_x*sin(angle)+placeLidar_y*cos(angle)*SimData.scale),
(int)(x_get*SimData.scale), (int)(y_get*SimData.scale) );
}
g2d.translate( -screen_pos_x, -screen_pos_y );
}

public void scan() {
    prevx = x.toArray(new Float[ x.size() ]);
    prevy = y.toArray(new Float[ y.size() ]);
    x.clear();
    y.clear();
    x_transformed.clear();
    y_transformed.clear();

    Random randGen = new Random();
    /// \todo parameter to allow clockwise or counterclockwise lidar
scans
    /// \todo ( placeLidar_x, placeLidar_y )
    //Point2D.Double CarOrigin = new Point2D.Double(
position_WC_x, -position_WC_y );
    Point2D.Float CarOrigin = new Point2D.Float(
(float)(position_WC_x+placeLidar_x), (float)(-position_WC_y+placeLidar_y) );
    for(radarAngle = radarAngleRangeFactor; radarAngle < (361f-
radarAngleRangeFactor); radarAngle += increment) {

```

```

        ArrayList<float[]> getSmallest = new ArrayList<float[]>();
        Line2D line = new Line2D.Float( CarOrigin, new
Point2D.Float((float)(CarOrigin.getX())+radarRange*cos(toRadians(radarAngle+90)+ang
le)),    //relative to car

        (float)(CarOrigin.getY())+radarRange*sin(toRadians(radarAngle+90)+angle)) );
        int i;
        int chker = 0;
        for(i = 0; i < chkList.size(); i++) {
            float[] tempOb = chkList.get(i);
            float[] xPoint = new float[tempOb.length/2];
            float[] yPoint = new float[tempOb.length/2];
            int j=0;
            int count = 0;
            for(;j<tempOb.length;) {
                xPoint[count] = tempOb[j];
                yPoint[count] = tempOb[j+1];
                j = j+2;
                count++;
            }
            float[] intersects =
Geometry.findLinePolygonIntersections(xPoint, yPoint, (float)line.getX2(),
(float)line.getY2(), (float)line.getX1(), (float)line.getY1());
            if(intersects != null) {
                getSmallest.add(intersects);
                chker = 1;
            }
        }
        float noise;
        /// \todo make settable from radial noise or (x-y) cartesian
noise
        float noiseMin = 0.0f;
        float noiseMax = 0.05f;
        noise = randGen.nextFloat()*noiseMax+noiseMin;
        // noise = 0;

        float xptAdd;
        float yptAdd;

        if(chker == 0) {
            xptAdd = (float)(line.getX2()-
CarOrigin.getX()+noise*cos(radarAngle*(PI/180)));
            yptAdd = (float)(line.getY2()-
CarOrigin.getY()+noise*sin(radarAngle*(PI/180)));
        }
        else {

```

```

        int index=0;
        float lengthcompare = (float)(sqrt(
    pow(line.getX2()-line.getX1(),2) + pow(line.getY2()-line.getY1(),2 )));
        for(i=0;i<getSmallest.size();i++) {
            float[] chkPts = getSmallest.get(i);
            float length = (float)(sqrt( pow( chkPts[0]-
    line.getX1(),2) + pow(chkPts[1]-line.getY1(),2 )));
            if(length<lengthcompare) {
                index = i;
                lengthcompare = length;
            }
        }
        float[] addPoints = getSmallest.get(index);
        xptAdd = (float)(addPoints[0]-
    CarOrigin.getX()+noise*cos(radarAngle*(PI/180)));
        yptAdd = (float)(addPoints[1]-
    CarOrigin.getY()+noise*sin(radarAngle*(PI/180)));
    }
    double rotateAngle = -(angle-PI/2);
    x.add(xptAdd);
    y.add(yptAdd);
    x_transformed.add((float)(xptAdd*cos(rotateAngle)-
    yptAdd*sin(rotateAngle)));
    y_transformed.add((float)(xptAdd*sin(rotateAngle)+yptAdd*cos(rotateAngle)));
}
this.record();
}
}
}

vdhils/Vehicle/CarParameters.java:
```

```

package vdhils.Vehicle;
//! @cond Doxygen_Suppress
public class CarParameters {
    // Model Properties
    private float DRAG=20f;                                ///< Drag coefficient
    private float RESISTANCE=0.1f;                            ///< Rolling resistance
    coefficient
    private float LOW_SPEED_THRES=1.25f;                   ///< Low speed under which
    the low speed Ackermann model is used to calculate forces
    // Geometry
    private float F2CG=0.212f;                             ///< Distance from CG to
    front tire (m)
```

```

        private float R2CG=0.115f;           ///< Distance from CG to rear
tire (m)
        private float h=0.05f;                ///< Distance from CG to
ground tire (m)
        private float length=0.61f;          ///< Car length (m)
        private float width=0.3f;            ///< Car width (m)
        private float wheeldiameter=0.1f;    ///< Car wheel's length (m)
        private float wheeldepth=0.05f;       ///< Car wheel's width (m)
// Mechanical Properties
        private float mass=4f;               ///< Car mass (kg)
        private float inertia=0.1f;          ///< Car inertia (kg*m^2)
        private float STEERLIMIT=17.0f;       ///< Mechanically imposed steering
limit
// Graphics Properties
        private float simScale = 0.2f; // < Car image to sprite scale in pixel per pixel
// Trail Parameters
        private int TRAIL_SIZE = 150;
        private float sensorRange = 21.0f;
        private float lidarScanRate = 0.05f;
        private float radarAngularRange = 240f;
        private int numberBeams=120;
        private boolean updateFromEncoders=false;
        private boolean align;
        private String roadway;
        private float roadWidth;
        private String carImage = "/Resources/Textures/Cars/sportscar-jag.png";
        private boolean linearTireModel = false;
        private String serialPort = "";
// Simulation Properties
        private float position_WC_x;
        private float position_WC_y;
        private float velocity_WC_x;
        private float velocity_WC_y;
        private float acceleration_WC_x;
        private float acceleration_WC_y;
        private float coneSize = 0.1f;
        private float coneSpacing = 1.22f;

        private float mew_s = 1.0f;             ///< Tire coefficient of friction (for
friction circle and traction limited model)
        private float k = 0.0f;                ///< Dynamic coefficient of friction
factor (sec/meter)
        private float CA_R = -50f;             ///< Rear Cornering Stiffness
(for linear model)
        private float CA_F = -50f;             ///< Front Cornering Stiffness
(for linear model)

```

```

private float B = 82f;                                ///< Stiffness Factor
private float C = 0.09f;                               ///< Shape Factor
private float D = 70f;                                 ///< Peak Factor
private float E = 0.65f;                               ///< Curvature Factor

private float SCSP;
private boolean enableSC;
private float Proportional = 100f;
private float Integral = 0.05f;
private float Derivative = 0.001f;

/*! Specifies the motor armature's resistance - used in the linear motor modeled*/
private float Ra = 0.00831f;
/*! Motor torque constant */
private float Kt = 0.0032f;
/*! Back EMF constant inverse */
private float Kv = 380f;
/*! Max applied motor voltage */
private float Ea = 15f;
/*! Effective gear ratio of motor transmission */
private float GearRatio = 0.1f;

private float angle;
private float angularvelocity;
private float angular_acceleration;
private float velocity_x;
private float velocity_y;

private float steerangle;
private float steerGearReduct = 12.0f;
private float throttle;
private float brake;
private float speed;

private float lfw = 3.0f;                             ///< Look ahead distance
private String Manuever = "None";
private boolean enableManuever;

public void setParameter(String param, String value) {
    if(param != null && !param.isEmpty() && value != null &&
    !value.isEmpty()) {
        switch(param.toLowerCase()) {
            case "manuever":
                setManuever(value);
            break;
            case "lfw":

```

```

        setLFW(Float.parseFloat(value));
break;
case "enable manuever":
    setEnableManuever(Boolean.parseBoolean(value));
break;
case "b":
    setB(Float.parseFloat(value));
break;
case "c":
    setC(Float.parseFloat(value));
break;
case "d":
    setD(Float.parseFloat(value));
break;
case "e":
    setE(Float.parseFloat(value));
break;
case "front cornering coefficient":
    setCA_F(Float.parseFloat(value));
break;
case "rear cornering coefficient":
    setCA_R(Float.parseFloat(value));
break;
case "front to cg":
    setF2CG(Float.parseFloat(value));
break;
case "rear to cg":
    setR2CG(Float.parseFloat(value));
break;
case "scsp":
    setSCSP(Float.parseFloat(value));
break;
case "proportional":
    setProportional(Float.parseFloat(value));
break;
case "integral":
    setIntegral(Float.parseFloat(value));
break;
case "derivative":
    setDerivative(Float.parseFloat(value));
break;
case "enable sc":
    setEnableSC(Boolean.parseBoolean(value));
break;
case "h":
    setH(Float.parseFloat(value));

```

```

break;
case "k":
    setK(Float.parseFloat(value));
break;
case "static friction coefficient":
    setMew(Float.parseFloat(value));
break;
case "length":
    setLength(Float.parseFloat(value));
break;
case "width":
    setWidth(Float.parseFloat(value));
break;
case "wheeldiameter":
    setWheeldiameter(Float.parseFloat(value));
break;
case "ra":
    setRa(Float.parseFloat(value));
break;
case "kt":
    setKt(Float.parseFloat(value));
break;
case "kv":
    setKv(Float.parseFloat(value));
break;
case "ea":
    setEa(Float.parseFloat(value));
break;
case "gearratio":
    setGearratio(Float.parseFloat(value));
break;
case "mass":
    setMass(Float.parseFloat(value));
break;
case "steer limit":
    setSteerlimit(Float.parseFloat(value));
break;
case "inertia":
    setInertia(Float.parseFloat(value));
break;
case "steer gear reduction":
    setSteerGearReduct(Float.parseFloat(value));
break;
case "drag":
    setDrag(Float.parseFloat(value));
break;

```

```

        case "resistance":
            setResistance(Float.parseFloat(value));
        break;
        case "low speed threshold":
            setLowSpeedThreshold(Float.parseFloat(value));
        break;
        case "sensor range":
            setSensorRange(Float.parseFloat(value));
        break;
        case "linear tire model":
            setLinearTireModel(Boolean.parseBoolean(value));
        break;
        case "update speed from encoders":

            setUpdateSpeedfromEncoders(Boolean.parseBoolean(value));
            break;
        case "global x-position":
            setXPos(Float.parseFloat(value));
        break;
        case "global y-position":
            setYPos(Float.parseFloat(value));
        break;
        case "yaw":
            setAngle(Float.parseFloat(value));
        break;
        case "angle":
            setAngle(Float.parseFloat(value));
        break;
        case "global x-velocity":

            break;
        case "global y-velocity":

            break;
        case "trail length":
            setTrailLength(Integer.parseInt(value));
        break;
        case "road width":
            setRoadWidth(Float.parseFloat(value));
        break;
        case "scale":
            setScale(Float.parseFloat(value));
        break;
        case "car image":
            setCarImage(value);
        break;

```

```

        case "align":
            setAlign(Boolean.parseBoolean(value));
        break;
        case "roadway":
            System.out.println("Setting roadway: "+value);
            setRoadway(value);
        break;
        case "serial port":
            setSerialPort(value);
        break;
        case "number of beams":
            setNumberofBeams(Integer.parseInt(value));
        break;
        case "scan angle":
            setScanAngle(Float.parseFloat(value));
        break;
        case "sensor frequency":
            setSensorFrequency(Float.parseFloat(value));
        break;
        case "cone size":
            setConeSize(Float.parseFloat(value));
        break;
        case "road cone spacing":
            setConeSpacing(Float.parseFloat(value));
        break;
    }
}
}

public boolean setConeSize(float coneSize) {
    if(coneSize > 0) {
        this.coneSize = coneSize;
        return true;
    } else
        return false;
}
public float getConeSize() {
    return coneSize;
}
public boolean setConeSpacing(float coneSpacing) {
    if(coneSpacing > 0) {
        this.coneSpacing = coneSpacing;
        return true;
    } else
        return false;
}
public float getConeSpacing() {

```

```

        return coneSpacing;
    }
    public boolean setTrailLength(int TRAIL_SIZE) {
        if(TRAIL_SIZE >= 0) {
            this.TRAIL_SIZE = TRAIL_SIZE;
            return true;
        } else
            return false;
    }
    public int getTrailLength() {
        return TRAIL_SIZE;
    }
    public boolean setRoadWidth(float roadWidth) {
        if(roadWidth >= 0.0f) {
            this.roadWidth = roadWidth;
            return true;
        } else
            return false;
    }
    public float getRoadWidth() {
        return roadWidth;
    }
    public void setNumberofBeams(int numberBeams) {
        this.numberBeams = numberBeams;
    }
    public int getNumberofBeams() {
        return numberBeams;
    }
    public boolean setScanAngle(float radarAngularRange) {
        if(radarAngularRange>=0.0f) {
            this.radarAngularRange = radarAngularRange;
            return true;
        } else
            return false;
    }
    public float getScanAngle() {
        return radarAngularRange;
    }
    public boolean setSensorFrequency(float lidarScanRate) {
        if(lidarScanRate>=0.0f) {
            this.lidarScanRate = lidarScanRate;
            return true;
        } else
            return false;
    }
    public float getSensorFrequency() {

```

```

        return lidarScanRate;
    }
    public void setEnableManuever(boolean enableManuever) {
        this.enableManuever = enableManuever;
    }
    public boolean getEnableManuever() {
        return enableManuever;
    }
    public void setManuever(String Manuever) {
        this.Manuever = Manuever;
    }
    public String getManuever() {
        return Manuever;
    }
    public boolean setLFW(float lfw) {
        if(lfw>=0f) {
            this.lfw = lfw;
            return true;
        } else
            return false;
    }
    public float getLFW() {
        return lfw;
    }
    public void setSerialPort(String serialPort) {
        this.serialPort = serialPort;
    }
    public String getSerialPort() {
        return serialPort;
    }
    public boolean setScale(float simScale) {
        if(simScale>0) {
            this.simScale = simScale;
            return true;
        } else
            return false;
    }
    public boolean setRa(float Ra) {
        if(Ra>=0f) {
            this.Ra = Ra;
            return true;
        } else
            return false;
    }
    public float getRa() {
        return Ra;
    }
}

```

```

    }
    public boolean setSCSP(float SCSP) {
        if(SCSP>=0f) {
            this.SCSP = SCSP;
            return true;
        } else
            return false;
    }
    public float getSCSP() {
        return SCSP;
    }
    public boolean setProportional(float Proportional) {
        if(Proportional>=0f) {
            this.Proportional = Proportional;
            return true;
        } else
            return false;
    }
    public float getProportional() {
        return Proportional;
    }
    public boolean setIntegral(float Integral) {
        if(Integral>=0f) {
            this.Integral = Integral;
            return true;
        } else
            return false;
    }
    public float getIntegral() {
        return Integral;
    }
    public boolean setDerivative(float Derivative) {
        if(Derivative>=0f) {
            this.Derivative = Derivative;
            return true;
        } else
            return false;
    }
    public float getDerivative() {
        return Derivative;
    }
    public void setEnableSC(boolean enableSC) {
        this.enableSC = enableSC;
    }
    public boolean getEnableSC() {
        return enableSC;
    }
}

```

```

    }
    public boolean setKt(float Kt) {
        if(Kt>=0f) {
            this.Kt = Kt;
            return true;
        } else
            return false;
    }
    public float getKt() {
        return Kt;
    }
    public boolean setKv(float Kv) {
        if(Kv>=0f) {
            this.Kv = Kv;
            return true;
        } else
            return false;
    }
    public float getKv() {
        return Kv;
    }
    public boolean setEa(float Ea) {
        if(Ea>=0f) {
            this.Ea = Ea;
            return true;
        } else
            return false;
    }
    public float getEa() {
        return Ea;
    }
    public boolean setGearratio(float GearRatio) {
        if(GearRatio>=0f) {
            this.GearRatio = GearRatio;
            return true;
        } else
            return false;
    }
    public float getGearratio() {
        // System.out.println("Gearratio: "+this.GearRatio);
        return GearRatio;
    }
    public float getScale() {
        return simScale;
    }
    public boolean setB(float B) {

```

```

        if(B>=0f) {
            this.B = B;
            return true;
        } else
            return false;
    }
    public float getB() {
        return B;
    }
    public boolean setC(float C) {
        if(C>=0f) {
            this.C = C;
            return true;
        } else
            return false;
    }
    public float getC() {
        return C;
    }
    public boolean setD(float D) {
        if(D>=0f) {
            this.D = D;
            return true;
        } else
            return false;
    }
    public float getD() {
        return D;
    }
    public boolean setE(float E) {
        if(E>=0f) {
            this.E = E;
            return true;
        } else
            return false;
    }
    public float getE() {
        return E;
    }
    public boolean setCA_F(float CA_F) {
        if(CA_F>=0f) {
            this.CA_F = CA_F;
            return true;
        } else
            return false;
    }
}

```

```

public float getCA_F() {
    return CA_F;
}
public boolean setCA_R(float CA_R) {
    if(CA_R>=0f) {
        this.CA_R = CA_R;
        return true;
    } else
        return false;
}
public float getCA_R() {
    return CA_R;
}
public boolean setK(float k) {
    if(k>=0f) {
        this.k = k;
        return true;
    } else
        return false;
}
public float getK() {
    return k;
}
public boolean setMew(float mew_s) {
    if(mew_s>=0f) {
        this.mew_s = mew_s;
        return true;
    } else
        return false;
}
public float getMew() {
    return mew_s;
}
public boolean setDrag(float DRAG) {
    if(DRAG>=0f) {
        this.DRAG = DRAG;
        return true;
    } else
        return false;
}
public float getDrag() {
    return DRAG;
}
public boolean setResistance(float RESISTANCE) {
    if(RESISTANCE>=0) {
        this.RESISTANCE = RESISTANCE;
    }
}

```

```

        return true;
    } else
        return false;
}
public float getResistance() {
    return RESISTANCE;
}
public boolean setSensorRange(float sensorRange) {
    if(sensorRange>0) {
        this.sensorRange = sensorRange;
        return true;
    } else
        return false;
}
public float getSensorRange() {
    return sensorRange;
}
public boolean setLowSpeedThreshold(float LOW_SPEED_THRES) {
    if(LOW_SPEED_THRES>=0) {
        this.LOW_SPEED_THRES = LOW_SPEED_THRES;
        return true;
    } else
        return false;
}
public float getLowspeedthres() {
    return LOW_SPEED_THRES;
}
public boolean setR2CG(float R2CG) {
    if(R2CG>0) {
        this.R2CG = R2CG;
        return true;
    } else
        return false;
}
public float getR2CG() {
    return R2CG;
}
public boolean setF2CG(float F2CG) {
    if(F2CG>0) {
        this.F2CG = F2CG;
        return true;
    } else
        return false;
}
public float getF2CG() {
    return F2CG;
}

```

```

    }
    public boolean setH(float h) {
        if(h>0) {
            this.h = h;
            return true;
        } else
            return false;
    }
    public float getH() {
        return h;
    }
    public boolean setLength(float length) {
        if(length>0) {
            this.length = length;
            return true;
        } else
            return false;
    }
    public float getLength() {
        return length;
    }
    public boolean setWidth(float width) {
        if(width>0) {
            this.width = width;
            return true;
        } else
            return false;
    }
    public float getWidth() {
        return width;
    }
    public boolean setWheeldiameter(float wheeldiameter) {
        if(wheeldiameter>0) {
            this.wheeldiameter = wheeldiameter;
            return true;
        }else
            return false;
    }
    public float getWheeldiameter() {
        return wheeldiameter;
    }
    public boolean setWheeldepth(float wheeldepth) {
        if(wheeldepth>0) {
            this.wheeldepth = wheeldepth;
            return true;
        }else

```

```

        return false;
    }
    public float getWheeldepth() {
        return wheeldepth;
    }
    public boolean setMass(float mass) {
        if(mass>0) {
            this.mass = mass;
            return true;
        }else
            return false;
    }
    public float getMass() {
        return mass;
    }
    public boolean setInertia(float inertia) {
        if(inertia>0) {
            this.inertia = inertia;
            return true;
        }else
            return false;
    }
    public float getInertia() {
        return inertia;
    }
    public boolean setSteerlimit(float steerlimit) {
        if(steerlimit>0) {
            this.STEERLIMIT = steerlimit;
            return true;
        }else
            return false;
    }
    public float getSteerlimit() {
        return STEERLIMIT;
    }
    public boolean setSteerGearReduct(float steerGearReduct) {
        if(steerGearReduct>0) {
            this.steerGearReduct = steerGearReduct;
            return true;
        } else
            return false;
    }
    public float getSteerGearReduct() {
        return steerGearReduct;
    }
    public void setUpdateSpeedfromEncoders(boolean updateFromEncoders) {

```

```

        this.updateFromEncoders = updateFromEncoders;
    }
    public boolean getUpdateSpeedfromEncoders() {
        return updateFromEncoders;
    }
    public void setXPos(float position_WC_x) {
        this.position_WC_x = position_WC_x;
    }
    public float getXPos() {
        return position_WC_x;
    }
    public void setYPos(float position_WC_y) {
        this.position_WC_y = position_WC_y;
    }
    public float getYPos() {
        return position_WC_y;
    }
    public void setAngle(float angle) {
        this.angle = angle;
    }
    public float getAngle() {
        return angle;
    }
    public void setAlign(boolean align) {
        this.align = align;
    }
    public boolean getAlign() {
        return align;
    }
    public void setRoadway(String roadway) {
        this.roadway = roadway;
    }
    public String getRoadway() {
        return roadway;
    }
    public void setLinearTireModel(boolean linearTireModel) {
        this.linearTireModel = linearTireModel;
    }
    public boolean isTireModelLinear() {
        return linearTireModel;
    }
    public void setCarImage(String fp) {
        carImage = fp;
    }
    public String getCarImage() {
        return carImage;
    }
}

```

```
    }
}

//! @endcond
```

vdhils/Vehicle/MotorModel.java:

```
/***
 *      \file MotorModel.java
 *      \brief MotorModel is a class that handles motor calculations for each car at each
 *      update.
 *              MotorModel.setState is first set and then motor parameters such as
 *      torque and speed
 *              can be queried.
 *
 *      References:
 *
 *      Revisions:
 *
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
*****
```

```

package vdhils.Vehicle;

import static java.lang.Math.*;

/** \class MotorModel
 * \brief MotorModel class handles updating of motor state based on selected motor
model (const or linear)
 */
public class MotorModel {
    // Motor Parameters
    /*! Specifies the constant force exerted by the motor when the constant force
motor model is used */
    protected float Const;
    /*! Specifies the motor armature's resistance - used in the linear motor modeled*/
    protected float Ra;           /* Traxxis: 0.00831; // (Ohms) */

    /*! Motor torque constant */
    protected float Kt;           /* Traxxis: 0.0032; */
    /*! Back EMF constant inverse */
    protected float Kv;           /* Traxxis: 350;      // (RPM/Volt) */
    /*! Max applied motor voltage */
    protected float Ea_max;        /* Traxxis: 14.29;     // (Volts) */
    /*! Effective gear ratio of motor transmission */
    protected float GearRatio;     /* Traxxis: 10;       // (-) */
    /*! Vehicle wheel radius */
    protected float R;             /* Traxxis: 0.10795;  //<
Wheel Radius (m) */

    /*! Angular velocity of motor shaft */
    protected double omega = 0;
    /*! Motor torque */
    protected double torque = 0;
    /*! Max tractive force applied by motor torque */
    protected double force_tractive = 0;
    /*! Instantaneous motor power output */
    protected double power = 0;
    /*! Instantaneous motor voltage (ideal power source assumed) */
    protected float voltage;
    /*! Instantaneous motor current */
    protected double current = 0;
    /*! Enum specifying which motor model to use when evaluating */
    protected MODEL model;

    public MotorModel() {
        Ra = 0.00831f;
        Kt = 0.0032f;
    }
}

```

```

        Kv =      350f;
        Ea_max =   14.29f;
        GearRatio = 1.0f;
        Const =     100.0f;
        R =         0.10795f;
        model =    MODEL.LINEAR;
        voltage =   14.29f; // Ideal battery no voltage sag across operating
range
    }
/*! Overloaded motor model constructor that accepts parameters for the linear tire
model
 * and sets the the motor model to user the linear equations
 * \param R      Vehicle weel radius
 * \param Ra     Motor aramture resistance
 * \param Kt     Motor torque constant
 * \param Kv     Back EMF constant inverse
 * \param Ea_max Max applied motor voltage
 */
public MotorModel(float R, float Ra, float Kt, float Kv, float Ea_max, float
GearRatio) {
    this.R = R;
    this.Ra = Ra;
    this.Kt = Kt;
    this.Kv = Kv;
    this.Ea_max = Ea_max;
    this.GearRatio = GearRatio;
    Const = 100.0f;
    model = MODEL.LINEAR;
    voltage = Ea_max;

    // System.out.println("Ra: "+this.Ra);
    // System.out.println("Kt: "+this.Kt);
    // System.out.println("Kv: "+this.Kv);
    // System.out.println("Ea: "+this.Ea_max);
}

public MotorModel(float R, float Const) {
    this.R = R;
    this.Const = Const;
    model = MODEL.CONST;
}

protected double getMotorTractiveForce() {
    return force_tractive;
}
/*! Get the motor torque

```

```

*      \return Motor torque
*/
protected double getMotorTorque() {
    return torque;
}

protected double getMotorTractiveTorque() {
    return (torque*GearRatio);
}
/*!
 *      Get the instantaneous motor power
 *      \return Instantaneous motor power
 */
protected double getMotorPower() {
    return power;
}
/*!
 *      Get the instantaneous motor current
 *      \return Instantaneous motor current
 */
protected double getMotorCurrent() {
    return current;
}

public void setMotorState(double longSpeed, float Throttle) {
    omega = GearRatio*R*longSpeed; // rad/s
                                                // Calculate motor angular speed
    // System.out.println("Omega: "+omega);
    if(model == MODEL.LINEAR) {
        torque = -
(Kt/(Ra*Kv))*omega+(Kt/Ra)*Ea_max*(Math.abs(Throttle)/100.0f);           //
Calculate motor torque
        torque *= signum(Throttle);
        if( (signum(Throttle)>0 && torque < 0) || (signum(Throttle)<0 &&
torque>0) ) {
            torque = 0;
        }
    }
    else {
        torque = Const*(R/GearRatio)*(Throttle/100.0f);
    }
    force_tractive = ((torque*GearRatio)/R);
    // System.out.println("force_tractive: "+force_tractive);
    power = torque*omega;
    current = power/voltage;
}

public void setConstMotorForce(float Const) {

```

```

        this.Const = Const;
    }
/*! This enum dictates which motor model equations which be used to
evaluate motor states */
public static enum MODEL {
    CONST, /*! Constant enum that applies a constant force through all
angular speeds */
    LINEAR /*! Linear enum that applies a linearly varying force with
angular speed */
};
}

```

vdhils/Vehicle/TireModel.java:

```

/*
 *      \file TireModel.java
 *      \brief TireModel contains an enum for selection of tire model and holds has an
inner class
 *              holding parameters for each model neccessary.
 *
 *      \author Thomas Stevens
 *
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package vdhils.Vehicle;
import static java.lang.Math.*;
/** \class TireModel
 * \brief TireModel contains an enum for selection of tire model and holds has an
inner class
 * holding parameters for each model neccessary.
*/
public class TireModel {
    protected float mew_s; ///< Tire coefficient of
friction (for friction circle and traction limited model)
    protected float k; ///< Dynamic coefficient of
friction factor (sec/meter)
    protected float CA_R; ///< Rear Cornering Stiffness (for
linear model)
    protected float CA_F; ///< Front Cornering Stiffness (for
linear model)
    protected float B; //< Stiffness Factor
    protected float C; //< Shape Factor
    protected float D; //< Peak Factor
    protected float E; //< Curvature Factor
    protected MODEL model; //< Enumeration to allow tire
model selection to be used during updates

    public TireModel() {
        mew_s = 0.72f;
        k = 0.012f;
        CA_R = -5.2f;
        CA_F = -5.2f;
        B = 0.1f; //0.00875f;
        C = 3f; //1000f;
        D = 130000f; //2.0f;
        E = 1.325f; //100000f;
        model = MODEL.MAGIC;
    }
    /*! Overloaded tire model constructor to that accepts Pacejka formula parameters
     * and sets the current tire model to use the Pacejka 'Magic' formula
     * \param B The Stiffness Factor
     * \param C The Shape Factor
     * \param D The Peak Factor
     * \param E The Curvature Factor
    */
    public TireModel(float B, float C, float D, float E) {
        mew_s = 0.72f;
        k = 0.012f;
        this.B = B;
        this.C = C;
    }
}

```

```

        this.D = D;
        this.E = E;
        model = MODEL.MAGIC;
    }
/*! Overloaded tire model constructor to that accepts linear tire model parameters
 * and sets the current tire model to use the linear tire model
 * \param CA_R      The rear tire cornering stiffness
 * \param CA_F      The front tire cornering stiffness
 */
public TireModel(float CA_R, float CA_F) {
    mew_s = 0.72f;
    k = 0.012f;
    this.CA_R = CA_R;
    this.CA_F = CA_F;
    model = MODEL.LINEAR;
}
/*! Calculates and returns the lateral force generated by the tire using the
 * specified tire model
 * \param slipangle   The vehicle sideslip angle
 * \return The lateral force generated by the tire
 */
public double getLateralForce(double slipangle) {
    // Set the model depending on which tire model is specified
    if(model == MODEL.LINEAR)
        return slipangle*CA_F;
    else
        return (-D*sin(C*atan(B*slipangle-E*(B*slipangle-
atan(B*slipangle))))));
}
/*! Calculates the dynamically impacted static friction coefficient as it varies
with speed
 * \param speed  Vehicle speed
 * \return The dynamically impacted static friction coefficient
*/
public float getDynamicFrictionCoeff(float speed) {
    // System.out.println("k: "+k);
    return mew_s*(1.0f-k*speed);
}
/** The MODEL enum dictates which tire model will be used during car
updates.
 */
public static enum MODEL {
    LINEAR, /*! Linear tire model where F = C*(slip) */
    MAGIC   /*! The Pacejka 'Magic Formula' Tire Model where F =
D*sin(C*arctan(B*(1-E)*slip+E*arctan(B*slip))) */
};

```

```

/*! Sets the tire model to use either a linear tire model or a Pacejka tire model
 * \param model Enum indicating which model to use
 */
public void setModel(MODEL model) {
    this.model = model;
}
/*! Sets the tires coefficient of static friction
 * \param mew_s The coefficient of static friction between the tire and the
road surface
*/
public void setStaticFrictionCoeff(float mew_s) {
    this.mew_s = mew_s;
}
/*! Set the dynamic coefficient of friction factor
 * \param k The dynamic factor coefficient which models the influence of the
vehicle's velocity on the static friction coefficient
*/
public void setDynamicFactorCoeff(float k) {
    this.k = k;
}

```

vdhils/GUI/MainMenu.java:

```

/**
* \file MainMenu.java
* \brief MainMenu contains GUI components related to manipulating SimData for
to influence SimUpdater object during creation.
*
*
* \author Thomas Stevens
*
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS

```

```

* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  

package vdhils.GUI;
// Import java API's/libraries
import vdhils.SimData;
// GUI Imports
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.Toolkit;
import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.border.EtchedBorder;  

/***
* \class MainMenu
* \brief MainMenu contains GUI components related to manipulating SimData for
to influence SimUpdater object during creation.
*/
public class MainMenu extends JPanel {  

    private ActionListener MainListener;
    private int width;
    private int height;
    private Color CalPoly_GOLD = new Color(122,91,17);
    private Color CalPoly_GREEN = new Color(2,73,48);
    private Color CalPoly_CREAM = new Color(214,204,175);
    private JButton startButton,exitButton,ModifyViewButton,SettingsButton;  

    public MainMenu(ActionListener listener,int width,int height,String versNumber) {
        MainListener = listener;
        this.width = width;
        this.height = height;  

        setLayout(new BorderLayout());
        GridLayout buttonLayout = new GridLayout(1,0);

```

```

        Border buttonBorder;
        Border blackLine = BorderFactory.createLineBorder(Color.black,3);
        buttonBorder = BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
        //Set up the horizontal gap value
buttonLayout.setHgap(25);
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(CalPoly_GOLD);
        buttonPanel.setBorder(buttonBorder);
        buttonPanel.setLayout(buttonLayout);

        createStartButton();
        createSettingsButton();
        createModifyViewButton();
        createExitButton();

        buttonPanel.add(startButton);
        buttonPanel.add(ModifyViewButton);
        buttonPanel.add(SettingsButton);
        buttonPanel.add(exitButton);

        add(new SplashScreen("/Resources/car.png",width,height,versNumber));

        add(buttonPanel,BorderLayout.PAGE_END);
setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

this.setOpaque(true);
buttonPanel.setOpaque(true);

        this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
        this.getActionMap().put("Escape pressed", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                System.exit(0);
            }
        });
    }

    @Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
    int w = getWidth();
}

```

```

        int h = getHeight();
        GradientPaint gradPaint = new GradientPaint(0,0,
CalPoly_GOLD,width,height, CalPoly_GREEN);
        g2d.setPaint(gradPaint);
        g2d.fillRect(0, 0, w, h);
    }

    private void createStartButton() {
        startButton = new JButton("START");
        startButton.setVerticalTextPosition(AbstractButton.CENTER);
        startButton.setHorizontalTextPosition(AbstractButton.CENTER);
        startButton.setMnemonic('S');
        startButton.setActionCommand("Start");
        startButton.addActionListener(MainListener);
        startButton.setBackground(CalPoly_CREAM);
        startButton.setToolTipText("Click this button to begin a new simulation");
    }

    private void createSettingsButton() {
        SettingsButton = new JButton("SETTINGS");
        SettingsButton.setVerticalTextPosition(AbstractButton.CENTER);
        SettingsButton.setHorizontalTextPosition(AbstractButton.CENTER);
        SettingsButton.setMnemonic('T');
        SettingsButton.setActionCommand("Settings");
        SettingsButton.addActionListener(MainListener);
        SettingsButton.setBackground(CalPoly_CREAM);
        SettingsButton.setToolTipText("Click this button to view and/or modify
simulation, graphics, controller, and misc. settings");
    }

    private void createModifyViewButton() {
        ModifyViewButton = new JButton("PARAMETERS");
        ModifyViewButton.setVerticalTextPosition(AbstractButton.CENTER);
        ModifyViewButton.setHorizontalTextPosition(AbstractButton.CENTER);
        ModifyViewButton.setMnemonic('M');
        ModifyViewButton.setActionCommand("Modify");
        ModifyViewButton.addActionListener(MainListener);
        ModifyViewButton.setBackground(CalPoly_CREAM);
    }

    private void createExitButton() {
        exitButton = new JButton("QUIT");
        exitButton.setVerticalTextPosition(AbstractButton.CENTER);
        exitButton.setHorizontalTextPosition(AbstractButton.CENTER);
        exitButton.setMnemonic('Q');
        exitButton.setActionCommand("Quit");
        exitButton.addActionListener(MainListener);
        exitButton.setBackground(CalPoly_CREAM);
    }
}

```

## vdhils/GUI/ParametersMenu.java:

```

/**
 * \file ParametersMenu.java
 * \brief SettingsMenu contains GUI components related to manipulating
 *        SimData to influence HardwareinloopSim object during creation.
 * \author Thomas Stevens
 *
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
*****  

*****  

package vdhils.GUI;  

import vdhils.SimData;  

import vdhils.Vehicle.CarParameters;  

//! Import Libraries  

import java.awt.*;  

import java.awt.event.*;  

import javax.swing.*;  

import javax.swing.event.*;  

import java.util.Vector;  

import javax.swing.table.AbstractTableModel;  

import java.awt.image.*;  

import java.awt.Toolkit;  

import javax.swing.border.Border;  

import javax.swing.border.TitledBorder;

```

```

import javax.swing.border.EtchedBorder;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;
import javax.swing.table.DefaultTableModel;
import javax.swing.event.TableModelEvent;
import jxl.Cell;
import jxl.Sheet;
import jxl.Workbook;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
// Serializable Import
import java.io.*;
import java.awt.Toolkit;

/// \todo Modify/View Car button button updating figure like carsim
public class ParametersMenu extends JPanel implements ActionListener,
TableModelListener, Serializable {

    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
    serialized output stream */
    private static final long serialVersionUID = 798907846110479146L;
    private ActionListener MainListener;
    private int width;
    private int height;
    private SimData simData;
    private JButton
    saveButton,exitButton,backButton,addObstacleButton,removeObstacleButton;
    private final JPanel parametersPanel = new JPanel();
    private ObstaclesTableModel obstaclesModel;
    private ParametersTableModel geometryModel;
    private ParametersTableModel modelModel;
    private ParametersTableModel graphicsModel;

    public ParametersMenu(ActionListener listener,int width,int height,SimData simData)
    {
        MainListener = listener;
        this.width = width;
        this.height = height;
        this.simData = simData;

        setLayout(new BorderLayout());

```

```

// setBackground(CalPoly_CREAM);

File parametersFile = new File("./VehicleParameters.xls");
geometryModel = new ParametersTableModel();
modelModel = new ParametersTableModel();
graphicsModel = new ParametersTableModel();
obstaclesModel = new ObstaclesTableModel();

// Get workbook from file
try {
    Workbook workBook = Workbook.getWorkbook(parametersFile);
    // Get geometry sheet from workbook
    Sheet geometrySheet = workBook.getSheet("Geometry & Mass");
    Sheet modelSheet = workBook.getSheet("Model");
    Sheet graphicsSheet = workBook.getSheet("Graphics");
    Sheet obstaclesSheet = workBook.getSheet("AI Vehicles");
    // get sheets column and row count
    int geometrySheetColumns = geometrySheet.getColumns();
    int geometrySheetRows = geometrySheet.getRows();
    int modelSheetColumns = modelSheet.getColumns();
    int modelSheetRows = modelSheet.getRows();
    int graphicsSheetColumns = graphicsSheet.getColumns();
    int graphicsSheetRows = graphicsSheet.getRows();
    int obstaclesSheetColumns = obstaclesSheet.getColumns();
    int obstaclesSheetRows = obstaclesSheet.getRows();

    //sets table's rows
    for (int i = 0; i < obstaclesSheetRows-1; i++) {
        Object[] data = new Object[obstaclesSheetColumns];
        obstaclesModel.insertData(data);
    }
    //Get cell contents and put them in the table
    for (int i = 1; i < obstaclesSheetRows; i++) {
        for (int j = 0; j < obstaclesSheetColumns; j++) {
            Cell cell = obstaclesSheet.getCell(j, i);
            obstaclesModel.setValueAt(cell.getContents(), i-1,
j);
        }
    }

    //sets table's rows
    for (int i = 0; i < geometrySheetRows-1; i++) {
        String[] data = new String[geometrySheetColumns];
        geometryModel.addRow(data);
    }
    //get Column name from the spreedsheet and set table's column
}

```

```

        for (int i = 0; i < geometrySheetColumns; i++) {
            Cell[] column = geometrySheet.getColumn(i);
            if (column.length > 0) {
                if (!column[0].getContents().equals("")) {
                    String columnName =
                        column[0].getContents();
                    geometryModel.addColumn(columnName);
                }
            }
        }
        //Get cell contents and put them in the table
        for (int i = 1; i < geometrySheetRows; i++) {
            for (int j = 0; j < geometrySheetColumns; j++) {
                Cell cell = geometrySheet.getCell(j, i);
                geometryModel.setValueAt(cell.getContents(), i-1,
j);
            }
        }

        //sets table's rows
        for (int i = 0; i < modelSheetRows-1; i++) {
            String[] data = new String[modelSheetColumns];
            modelModel.addRow(data);
        }
        //get Column name from the spreedsheet and set table's column
        for (int i = 0; i < modelSheetColumns; i++) {
            Cell[] column = modelSheet.getColumn(i);
            if (column.length > 0) {
                if (!column[0].getContents().equals("")) {
                    String columnName =
                        column[0].getContents();
                    modelModel.addColumn(columnName);
                }
            }
        }
        //Get cell contents and put them in the table
        for (int i = 1; i < modelSheetRows; i++) {
            for (int j = 0; j < modelSheetColumns; j++) {
                Cell cell = modelSheet.getCell(j, i);
                modelModel.setValueAt(cell.getContents(), i-1, j);
            }
        }

        //sets table's rows
        for (int i = 0; i < graphicsSheetRows-1; i++) {
            String[] data = new String[graphicsSheetColumns];
            graphicsModel.addRow(data);
        }
    }
}

```

```

        }
        //get Column name from the spreedsheet and set table's column
        for (int i = 0; i < graphicsSheetColumns; i++) {
            Cell[] column = graphicsSheet.getColumn(i);
            if (column.length > 0) {
                if (!column[0].getContents().equals("")) {
                    String columnName =
                    column[0].getContents();
                    graphicsModel.addColumn(columnName);
                }
            }
        }
        //Get cell contents and put them in the table
        for (int i = 1; i < graphicsSheetRows; i++) {
            for (int j = 0; j < graphicsSheetColumns; j++) {
                Cell cell = graphicsSheet.getCell(j, i);
                graphicsModel.setValueAt(cell.getContents(), i-1,
                j);
            }
        }
    } catch(Exception ex) {}

    JTable geometryTable = new JTable();
    geometryTable.setModel(geometryModel);
    geometryTable.setPreferredScrollableViewportSize( new
    Dimension(width-100,height/3-150) );
    geometryTable.getModel().addTableModelListener(this);
    geometryTable.setBackground(Color.white);

    JTable modelTable = new JTable();
    modelTable.setModel(modelModel);
    modelTable.setPreferredScrollableViewportSize( new Dimension(width-
    100,height/3-150) );
    modelTable.getModel().addTableModelListener(this);
    modelTable.setBackground(Color.white);

    JTable graphicsTable = new JTable();
    graphicsTable.setModel(graphicsModel);
    graphicsTable.setPreferredScrollableViewportSize( new
    Dimension(width-100,height/3-150) );
    graphicsTable.getModel().addTableModelListener(this);
    graphicsTable.setBackground(Color.white);

    JTable obstacleTable = new JTable();
    obstacleTable.setModel(obstaclesModel);

```

```

        obstacleTable.setPreferredScrollableViewportSize( new
Dimension(width-100,height/3-175) );
        obstacleTable.getModel().addTableModelListener(new
TableModelListener() {
            @Override
            public void tableChanged(TableModelEvent e) {
                if(e.getType() == TableModelEvent.UPDATE)
                    System.out.println("Update");
                else if(e.getType() == TableModelEvent.INSERT)
                    System.out.println("Insert");
                int row = e.getFirstRow();
                int column = e.getColumn();
                if(column!=TableModelEvent.ALL_COLUMNS) {
                    ObstaclesTableModel model =
(ObstaclesTableModel)e.getSource();
                    //String columnName =
model.getColumnName(column);
                    Object data = model.getValueAt(row, column);

                    // Write data to xcel spreadsheet
                }
            }
        });
        obstacleTable.setBackground(Color.white);
        // obstacleTable.setOpaque(false);

        // setUpManueverColumn(obstacleTable,
obstacleTable.getColumnModel().getColumn(obstacleTable.getColumnCount()-2));
        // setUpAlignColumn(obstacleTable,
obstacleTable.getColumnModel().getColumn(obstacleTable.getColumnCount()-1));

        //Create the scroll pane and add the table to it.
JScrollPane geometryScrollPane = new JScrollPane();
geometryScrollPane.setViewportView(geometryTable);
//Create the scroll pane and add the table to it.
JScrollPane modelScrollPane = new JScrollPane();
modelScrollPane.setViewportView(modelTable);
//Create the scroll pane and add the table to it.
JScrollPane graphicsScrollPane = new JScrollPane();
graphicsScrollPane.setViewportView(graphicsTable);
//Create the scroll pane and add the table to it.
JScrollPane obstacleScrollPane = new JScrollPane();
obstacleScrollPane.setViewportView(obstacleTable);
// obstacleScrollPane.setOpaque(false);

```

```

        //Set up column sizes.
initColumnSizes(geometryTable);
    initColumnSizes(modelTable);
    initColumnSizes(graphicsTable);
    initColumnSizes(obstacleTable);

    GridLayout buttonLayout = new GridLayout(1,0);
    //Set up the horizontal gap value
buttonLayout.setHgap(25);
    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(simData.CalPoly_GOLD);
    Border buttonBorder;
    Border blackLine = BorderFactory.createLineBorder(Color.black,3);
    buttonBorder = BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
    buttonPanel.setBorder(buttonBorder);
    buttonPanel.setLayout(buttonLayout);

    createSaveButton();
    createBackButton();
    createExitButton();
    createAddObstacleButton();
    createRemoveObstacleButton();

    buttonPanel.add(saveButton);
    buttonPanel.add(addObstacleButton);
    buttonPanel.add(removeObstacleButton);
    buttonPanel.add(backButton);
    buttonPanel.add(exitButton);

    JPanel geometryTableHolder = new JPanel();
    geometryTableHolder.setBackground(simData.CalPoly_CREAM);
    // geometryTableHolder.setPreferredSize(new Dimension( width-100,
height/3-125 ));
    TitledBorder geometryHolderBorder =
BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
    "Vehicle Geometry Parameters:");
    geometryHolderBorder.setTitleJustification(TitledBorder.CENTER);
    geometryTableHolder.setBorder(geometryHolderBorder);
    geometryTableHolder.add(geometryScrollPane);

    JPanel modelTableHolder = new JPanel();
    modelTableHolder.setBackground(simData.CalPoly_CREAM);
    TitledBorder modelHolderBorder =
BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
    "Vehicle Model Parameters:");

```

```

modelHolderBorder.setTitleJustification(TitledBorder.CENTER);
modelTableHolder.setBorder(modelHolderBorder);
modelTableHolder.add(modelScrollPane);

JPanel graphicsTableHolder = new JPanel();
graphicsTableHolder.setBackground(simData.CalPoly_CREAM);
TitledBorder graphicslHolderBorder =
BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
                                "Vehicle Graphics Parameters:");
graphicslHolderBorder.setTitleJustification(TitledBorder.CENTER);
graphicsTableHolder.setBorder(graphicslHolderBorder);
graphicsTableHolder.add(graphicsScrollPane);

JPanel obstacleTableHolder = new JPanel();
obstacleTableHolder.setBackground(simData.CalPoly_CREAM);
TitledBorder obstacleHolderBorder =
BorderFactory.createTitledBorder(BorderFactory.createEmptyBorder(),
                                "AI Vehicle Parameters:");
obstacleHolderBorder.setTitleJustification(TitledBorder.CENTER);
obstacleTableHolder.setBorder(obstacleHolderBorder);
obstacleTableHolder.add(obstacleScrollPane);

parametersPanel.setLayout(new GridLayout(0,1));
parametersPanel.setBorder(blackLine);
parametersPanel.setBackground(simData.CalPoly_CREAM);
parametersPanel.add(geometryTableHolder);
parametersPanel.add(modelTableHolder);
parametersPanel.add(graphicsTableHolder);
parametersPanel.add(obstacleTableHolder);

add(parametersPanel, BorderLayout.CENTER);
add(buttonPanel, BorderLayout.PAGE_END);
setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
this.getActionMap().put("Escape pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        System.exit(0);
    }
});
}

public CarParameters getCarParameters() {

```

```

        CarParameters carParams = new CarParameters();

        for (int i = 0; i < geometryModel.getRowCount(); i++) {
            String param = (String)geometryModel.getValueAt( i, 0
).toString().toLowerCase();
            //System.out.println("Param: "+param);
            if(param != null && !param.isEmpty())

                carParams.setParameter(param,(String)geometryModel.getValueAt( i,
geometryModel.getColumnCount()-1 ).toString());
        }
        //Get cell contents and put them in the table
        for (int i = 0; i < modelModel.getRowCount(); i++) {
            String param = (String)modelModel.getValueAt( i, 0
).toString().toLowerCase();
            //System.out.println("Param: "+param);
            if(param != null && !param.isEmpty())
                carParams.setParameter(param,modelModel.getValueAt( i,
modelModel.getColumnCount()-1 ).toString());
        }
        //Get cell contents and put them in the table
        for (int i = 0; i < graphicsModel.getRowCount(); i++) {
            String param = (String)graphicsModel.getValueAt( i, 0
).toString().toLowerCase();
            //System.out.println("Param: "+param);
            if(param != null && !param.isEmpty())
                carParams.setParameter(param,graphicsModel.getValueAt(
i, graphicsModel.getColumnCount()-1 ).toString());
        }

        return carParams;
    }
    public ArrayList<CarParameters> getObstacleParameters() {
        ArrayList<CarParameters> obstacleParams = new
ArrayList<CarParameters>();

        //Get cell contents and put them in the table
        for (int i = 0; i < obstaclesModel.getRowCount(); i++) {
            CarParameters tempParams = new CarParameters();
            for(int j=0; j<obstaclesModel.getColumnCount(); j++) {
                String param =
obstaclesModel.getColumnName(j).toLowerCase();
                // System.out.println("Param: "+param);
                if(param != null && !param.isEmpty())

                    tempParams.setParameter(param,obstaclesModel.getValueAt( i, j ).toString());
            }
        }
    }
}

```

```

        }

        obstacleParams.add(tempParams);
    }

    return obstacleParams;
}

/*
 * This method picks good column sizes.
 * If all column heads are wider than the column's cells'
 * contents, then you can just use column.sizeWidthToFit().
 */
private void initColumnSizes(JTable table) {

    try {
        ParametersTableModel model =
(ParametersTableModel)table.getModel();
        TableColumn column = null;
        Component comp = null;
        int headerWidth = 0;
        int cellWidth = 0;
        Object[] longValues = model.longValues;
        TableCellRenderer headerRenderer =
table.getTableHeader().getDefaultRenderer();

        for (int i = 0; i < table.getColumnCount(); i++) {
            column = table.getColumnModel().getColumn(i);

            comp = headerRenderer.getTableCellRendererComponent(
null, column.getHeaderValue(),
false, false, 0, 0);
            headerWidth = comp.getPreferredSize().width;

            comp =
table.getDefaultRenderer(model.getColumnClass(i)).
getTableCellRendererComponent(
table, longValues[i],
false, false, 0, i);
            cellWidth = comp.getPreferredSize().width;

            column.setPreferredWidth(Math.max(headerWidth,
cellWidth));
        }
    }
    catch(Exception ClassCastException) {

```

```

        ObstaclesTableModel model =
(ObstaclesTableModel)table.getModel();
        TableColumn column = null;
        Component comp = null;
        int headerWidth = 0;
        int cellWidth = 0;
        Object[] longValues = model.longValues;
        TableCellRenderer headerRenderer =
table.getTableHeader().getDefaultRenderer();

        for (int i = 0; i < table.getColumnCount(); i++) {
            column = table.getColumnModel().getColumn(i);

            comp = headerRenderer.getTableCellRendererComponent(
null, column.getHeaderValue(),
false, false, 0, 0);
            headerWidth = comp.getPreferredSize().width;

            cellWidth = comp.getPreferredSize().width;

            column.setPreferredWidth(Math.max(headerWidth,
cellWidth));
        }
    }

private class ParametersTableModel extends AbstractTableModel {

    public Object[] longValues = { "LOW_SPEED_THRES",
        "Max lateral force at optimum slip angle (the max available grip or
the tire friction circle radius)",
        "kg*m^2",
        "Boolean",
        "100000.0"
    };
    public List<String> tableColumnNames = new ArrayList<String>();
    private ArrayList<Object[]> data = new ArrayList<Object[]>();

    public ParametersTableModel() {

    }

    public int getColumnCount() {
        return tableColumnNames.size();
    }
}

```

```

public int getRowCount() {
    return data.size();
}

public String getColumnNames(int col) {
    return tableColumnNames.get(col);
}

public Object getValueAt(int row, int col) {
    return data.get(row)[col];
}

public Class getColumnClass(int c) {
    return getValueAt(data.size()-1, c).getClass();
}

public boolean isCellEditable(int row, int col) {
    // Note that the data/cell address is constant,
    // no matter where the cell appears onscreen.
    if (col < tableColumnNames.size()-1) {
        return false;
    } else {
        return true;
    }
}

public void setValueAt(Object value, int row, int col) {
    data.get(row)[col] = (String)value;
    this.fireTableCellUpdated(row, col);
}

public void addRow(String[] row) {
    data.add(row);
    int size = data.size();
    this.fireTableRowsInserted(size, size);
}

public void addColumn(String columnString) {
    tableColumnNames.add(columnString);
}

public void delRow(int row) {
    data.remove(row);
    //this.fireTableRowsDeleted();
}

```

```

    }

    private void setUpManueverColumn(JTable table, TableColumn
manueverColumn) {
    //Set up the editor for the sport cells.
    JComboBox comboBox = new JComboBox();
    DefaultComboBoxModel c_model = new DefaultComboBoxModel();
    c_model.addElement("None");
    c_model.addElement("SLC");
    c_model.addElement("DLC");
    c_model.addElement("Center");
    comboBox.setModel(c_model);
    manueverColumn.setCellEditor(new DefaultCellEditor(comboBox));

    c_model = new DefaultComboBoxModel();
    c_model.addElement("None");
    c_model.addElement("SLC");
    c_model.addElement("DLC");
    c_model.addElement("Center");
    //Set up tool tips for the sport cells.
    ComboBoxTableCellRenderer renderer = new ComboBoxTableCellRenderer();
    // renderer.setToolTipText("Click for combo box");
    renderer.setModel(c_model);
    manueverColumn.setCellRenderer(renderer);
}

private void setUpAlignColumn(JTable table, TableColumn col) {
//Set up the editor for the sport cells.
JCheckBox checkBox = new JCheckBox();

col.setCellEditor(new DefaultCellEditor(checkBox));

//Set up tool tips for the sport cells.
DefaultTableCellRenderer renderer = new DefaultTableCellRenderer();
renderer.setToolTipText("Click for check box");
col.setCellRenderer(renderer);
}

public class ComboBoxTableCellRenderer extends JComboBox implements
TableCellRenderer {

    @Override
    public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int column) {
        setSelectedItem(value);
        return this;
}

```

```

    }

}

private class ObstaclesTableModel extends AbstractTableModel {
    private String[] columnNames = {"Vehicle ID", "Mass", "Inertia", "a",
        "b", "h", "Length", "Width", "WheelDiameter", "Global X-Position", "Global Y-
        Position", "Angle", "SCSP", "Enable SC", "Manuever", "Enable Manuever", "Align"};
    private Vector data = new Vector();

    public final Object[] longValues = {new Integer(20), new Float(100000),
        new Float(100000), new Float(100), new Float(100), new Float(100), new Float(100),
        new Float(100), new Float(100), new Float(10000), new Float(10000), new Float(10),
        "Manuever", new Boolean(false)};

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public int getRowCount() {
        return data.size();
    }

    @Override
    public Object getValueAt(int row, int col) {
        return ((Vector) data.get(row)).get(col);
    }

    public String getColumnName(int col){
        return columnNames[col];
    }
    public Class getColumnClass(int c){
        return getValueAt(0,c).getClass();
    }

    public void setValueAt(Object value, int row, int col){
        ((Vector) data.get(row)).setElementAt(value, col);
        fireTableCellUpdated(row,col);
    }

    public boolean isCellEditable(int row, int col){
        return true;
    }
}

```

```

public void insertData(Object[] values){
    data.add(new Vector());
    for(int i =0; i<values.length; i++){
        if(i==0)
            ((Vector) data.get(data.size()-1)).add(data.size());
        else
            ((Vector) data.get(data.size()-1)).add(values[i]);
    }
    fireTableDataChanged();
}

public void removeRow(int row){
    data.removeElementAt(row);
    fireTableDataChanged();
}

public void removeRow(){
    data.removeElementAt(data.size()-1);
    fireTableDataChanged();
}
}

private boolean writeToExcel() {
    // WritableWorkbook writWorkbook = Workbook.createWorkbook(new
File("Output.xls"));
    // WritableSheet wrtsheet0 = writWorkbook.createSheet("Geometry &
Mass",0);
    return false;
}
@Override
public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();
    // System.out.println(action);
    if(action.equals("Save")) {
        // createJFileChooser()://"Save"
    }
    else if(action.equals("ADD")) {
        Object[] tempOb =
{1,3.117,0.1,0.212,0.115,0.051,0.61,0.3,0.1,0.0,0.0,0.0,0.0, "false", "None", "false",
"false" }; //Boolean.FALSE
        obstaclesModel.insertData(tempOb);
    }
    else if(action.equals("REMOVE")) {
        obstaclesModel.removeRow();
    }
}

```

```

        }
    @Override
    public void tableChanged(TableModelEvent e) {
        if(e.getType() == TableModelEvent.UPDATE)
            System.out.println("Update");
        else if(e.getType() == TableModelEvent.INSERT)
            System.out.println("Insert");
        int row = e.getFirstRow();
        int column = e.getColumn();
        ParametersTableModel model = (ParametersTableModel)e.getSource();
        String columnName = model.getColumnName(column);
        Object data = model.getValueAt(row, column);
        // Write data to xcel spreadsheet
    }
    private void createJFileChooser() {
        final JFileChooser fileChooser = new JFileChooser();
        ActionListener fileListener = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                //Process action performed
                String action = (String) ae.getActionCommand();
                if( action == "ApproveSelection")
                    System.out.println( (String)
fileChooser.getSelectedFile().getAbsolutePath() );
                remove( fileChooser );
                add( parametersPanel );
                validate();
            }
        };
        fileChooser.addActionListener( fileListener );
        remove(parametersPanel);
        add( fileChooser );
        validate();
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
        int w = getWidth();
        int h = getHeight();

```

```

        GradientPaint gradPaint = new GradientPaint(0, 0,
simData.CalPoly_GOLD, width, height, simData.CalPoly_GREEN);
        g2d.setPaint(gradPaint);
        g2d.fillRect(0, 0, w, h);
    }

    private void createSaveButton() {
        saveButton = new JButton("SAVE");
        saveButton.setVerticalTextPosition(AbstractButton.CENTER);
        saveButton.setHorizontalTextPosition(AbstractButton.CENTER);
        saveButton.setMnemonic('s');
        saveButton.setActionCommand("Save");
        saveButton.addActionListener(this);
        saveButton.setBackground(simData.CalPoly_CREAM);

        saveButton.setEnabled(false);
    }

    private void createBackButton() {
        backButton = new JButton("BACK");
        backButton.setVerticalTextPosition(AbstractButton.CENTER);
        backButton.setHorizontalTextPosition(AbstractButton.CENTER);
        backButton.setMnemonic('B');
        backButton.setActionCommand("BackFromParameters");
        backButton.addActionListener(MainListener);
        backButton.setBackground(simData.CalPoly_CREAM);
    }

    private void createExitButton() {
        exitButton = new JButton("QUIT");
        exitButton.setVerticalTextPosition(AbstractButton.CENTER);
        exitButton.setHorizontalTextPosition(AbstractButton.CENTER);
        exitButton.setMnemonic('Q');
        exitButton.setActionCommand("Quit");
        exitButton.addActionListener(MainListener);
        exitButton.setBackground(simData.CalPoly_CREAM);
    }

    private void createAddObstacleButton() {
        addObstacleButton = new JButton("ADD OBSTACLE");
        addObstacleButton.setVerticalTextPosition(AbstractButton.CENTER);
        addObstacleButton.setHorizontalTextPosition(AbstractButton.CENTER);
        addObstacleButton.setMnemonic('A');
        addObstacleButton.setActionCommand("ADD");
        addObstacleButton.addActionListener(this);
        addObstacleButton.setBackground(simData.CalPoly_CREAM);
    }

    private void createRemoveObstacleButton() {
        removeObstacleButton = new JButton("REMOVE OBSTACLE");

```

```

        removeObstacleButton.setVerticalTextPosition(AbstractButton.CENTER);

        removeObstacleButton.setHorizontalTextPosition(AbstractButton.CENTER);
        removeObstacleButton.setMnemonic('R');
        removeObstacleButton.setActionCommand("REMOVE");
        removeObstacleButton.addActionListener(this);
        removeObstacleButton.setBackground(simData.CalPoly_CREAM);
    }
}

```

vdhils/GUI/SettingsMenu.java:

```

/*
 *      \file SettingsMenu.java
 *
 *      \author Thomas Stevens
 *
 *      SettingsMenu contains GUI components related to manipulating
 *      SimData to influence HardwareinloopSim object during creation.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.*/
//*****
*****  

package vdhils.GUI;
import vdhils.SimData;
import vdhils.SettingsData;
```

```

//! Import Libraries
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.image.*;
import java.awt.Toolkit;
import javax.swing.border.Border;
import javax.swing.border.TitledBorder;
import javax.swing.border.EtchedBorder;

// Modify/View Car button
public class SettingsMenu extends JPanel implements ActionListener {

    private ActionListener MainListener;
    private int width;
    private int height;
    private SimData simData;
    private Color CalPoly_GOLD = new Color(122,91,17);
    private Color CalPoly_GREEN = new Color(2,73,48);
    private Color CalPoly_CREAM = new Color(214,204,175);
    private JButton saveButton,exitButton,backButton,loadButton;
    private final JTabbedPane pane = new JTabbedPane();

    public SettingsMenu(ActionListener listener,int width,int height,SimData simData) {
        MainListener = listener;
        this.width = width;
        this.height = height;
        this.simData = simData;

        setLayout(new BorderLayout());

        pane.add("Graphics
Settings",this.simData.getSettingsData().getGraphicsSettings());
        pane.add("Input
Settings",this.simData.getSettingsData().getJoystickSettings());
        // pane.add("Simulation
Settings",this.simData.getSettingsData().getSimulationSettings());
        pane.add("Embedded Device
Settings",this.simData.getSettingsData().getEmbeddedSettings());

        GridLayout buttonLayout = new GridLayout(1,0);
        //Set up the horizontal gap value
        buttonLayout.setHgap(25);
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(CalPoly_GOLD);
        Border buttonBorder;

```

```

        Border blackLine = BorderFactory.createLineBorder(Color.black,3);
        buttonBorder = BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
        buttonPanel.setBorder(buttonBorder);
        buttonPanel.setLayout(buttonLayout);

        createSaveButton();
        createLoadButton();
        createBackButton();
        createExitButton();

        buttonPanel.add(saveButton);
        buttonPanel.add(loadButton);
        buttonPanel.add(backButton);
        buttonPanel.add(exitButton);

        pane.setBorder(blackLine);
        pane.setBackground(CalPoly_CREAM);

        add(pane,BorderLayout.CENTER);
        add(buttonPanel,BorderLayout.PAGE_END);
        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

        this.setOpaque(true);
        buttonPanel.setOpaque(true);

        this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
        this.getActionMap().put("Escape pressed", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                System.exit(0);
            }
        });
    }

    public String getSerialPort() {
        SettingsData.EmbeddedSettings tempSettings =
this.simData.getSettingsData().getEmbeddedSettings();
        return tempSettings.getSerialPort();
    }

    public void actionPerformed(ActionEvent e) {
        String action = (String) e.getActionCommand();
        if(action.equals("Save")) {
            createJFileChooser();
        }
    }
}

```

```

        else if(action.equals("Load")) {
            createJFileChooser();
        }
    }
    private void createJFileChooser() {
        final JFileChooser fileChooser = new JFileChooser();
        ActionListener fileListener = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {
                //Process action performed
                String action = (String) ae.getActionCommand();
                // if( action == "ApproveSelection")
                // System.out.println( (String)
                fileChooser.getSelectedFile().getAbsolutePath() );

                remove( fileChooser );
                add(pane);
                validate();
            }
        };
        fileChooser.addActionListener( fileListener );

        remove( pane );
        add( fileChooser );
        validate();
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
        RenderingHints.VALUE_RENDER_QUALITY);
        int w = getWidth();
        int h = getHeight();
        GradientPaint gradPaint = new GradientPaint(0,0,
        CalPoly_GOLD,width,height, CalPoly_GREEN);
        g2d.setPaint(gradPaint);
        g2d.fillRect(0, 0, w, h);
    }
    private void createSaveButton() {
        saveButton = new JButton("SAVE");
        saveButton.setVerticalTextPosition(AbstractButton.CENTER);
        saveButton.setHorizontalTextPosition(AbstractButton.CENTER);
        saveButton.setMnemonic('s');
        saveButton.setActionCommand("Save");
        saveButton.addActionListener(this);
    }
}

```

```

        saveButton.setBackground(CalPoly_CREAM);

        saveButton.setEnabled(false);
    }
    private void createLoadButton() {
        loadButton = new JButton("LOAD");
        loadButton.setVerticalTextPosition(AbstractButton.CENTER);
        loadButton.setHorizontalTextPosition(AbstractButton.CENTER);
        loadButton.setMnemonic('L');
        loadButton.setActionCommand("Load");
        loadButton.addActionListener(this);
        loadButton.setBackground(CalPoly_CREAM);

        loadButton.setEnabled(false);
    }
    private void createBackButton() {
        backButton = new JButton("BACK");
        backButton.setVerticalTextPosition(AbstractButton.CENTER);
        backButton.setHorizontalTextPosition(AbstractButton.CENTER);
        backButton.setMnemonic('B');
        backButton.setActionCommand("BackFromSettings");
        backButton.addActionListener(MainListener);
        backButton.setBackground(CalPoly_CREAM);
    }
    private void createExitButton() {
        exitButton = new JButton("QUIT");
        exitButton.setVerticalTextPosition(AbstractButton.CENTER);
        exitButton.setHorizontalTextPosition(AbstractButton.CENTER);
        exitButton.setMnemonic('Q');
        exitButton.setActionCommand("Quit");
        exitButton.addActionListener(MainListener);
        exitButton.setBackground(CalPoly_CREAM);
    }
}

```

vdhils/GUI/SplashScreen.java:

```

/**
 *      \file SplashScreen.java
 *      \brief Splash screen contains GUI components to render splash screen's image
 *
 *      \author Thomas Stevens
 *
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"

```

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  
 LIMITED TO, THE  
 \* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
 PARTICULAR PURPOSE  
 \* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  
 CONTRIBUTORS BE  
 \* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
 EXEMPLARY, OR CONSEQUEN-  
 \* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
 SUBSTITUTE GOODS  
 \* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
 INTERRUPTION) HOWEVER  
 \* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,  
 STRICT LIABILITY,  
 \* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY  
 WAY OUT OF THE USE  
 \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
 DAMAGE. \*/

```

//*****
*****  

package vdhils.GUI;  

import vdhils.Graphics.Sprite;  

import vdhils.SimData;  

//! Import Libraries  

import java.awt.*;  

import javax.swing.*;  

import java.awt.image.*;  

/** \class SplashScreen  

 * \brief SplashScreen class loads, filters, and draws splash screen  

 */  

public class SplashScreen extends JPanel {  

    private int width;  

    private int height;  

    private Image image;  

    private String versNumber;  

    public SplashScreen(String filepath,int width,int height,String versNumber) {  

        this.width = width;  

        this.height = height;  

        this.versNumber = versNumber;  

        java.net.URL imgURL = getClass().getResource(filepath);  

        BufferedImage mImage;  

        try{  

            mImage = javax.imageio.ImageIO.read(imgURL);  

            int color = mImage.getRGB(0, 0);
  
```

```

        image = Sprite.makeColorTransparent(mImage, new Color(color));
    } catch(Exception e) {
        System.out.println("Pic error");
        SimData.logger.warning("Failed to load sprite (" + e.toString() +
    ")");
    }

    this.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    this.setOpaque(false);
}

public void drawBackground(Graphics2D g) {
    g.drawImage( image,0,0,width,(int)(height*0.85), this);
    g.setFont(new Font("Copperplate Gothic Bold", Font.PLAIN, 40));
    g.setColor(Color.white);
    g.drawString("VDHILS - " + versNumber, 100, 40);
    g.setFont(new Font("Copperplate Gothic Bold", Font.PLAIN, 30));
    g.drawString("Vehicle Dynamics Hardware in the Loop Simulator", 100,
75);
}
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    drawBackground(g2d);
}
}

```

vdhils/Graphics/SimRenderer.java:

```

/**
 *      \file SimRenderer.java
 *      \brief SimRenderer handles the high level rendering of the sim at the
BufferStrategy level, as well as handles updating time based events and animations.
*
*      References:
*
*      Revisions:
*          \li 4/6/2013 TFS
*
*      License:
*      This file is copyright 2014 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/

```

```

/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package vdhils.Graphics;
// Import java API's/libraries
import vdhils.SimData;
import vdhils.GUI.*;

// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, and other utilities)
import java.util.Random;
import java.util.concurrent.*;
import java.util.*;
import java.text.*;
// import javax.imageio.*;
```

```

// import java.io.*;

/**
 *      \class SimRenderer
 *          \brief SimRenderer handles the high level rendering of the sim at the
BufferStrategy level
 *
 */
public class SimRenderer implements ResizeListener {
    private SimData simData;
    private BlockingQueue<BufferStrategy> bufferStrategyQueue;
    // Not initialized at creation, but passed in externally when created
    private BufferStrategy bufferStrategy;
    // The component to draw via EDT
    private Component componentToDraw;
    private final Rectangle drawAreaBounds;
        // private int imageRendered = 0;
    // We draw almost all graphics to this image, then stretch it over the
    // entire frame.
    // This allows a resize to make the sim bigger, as opposed to
    // just providing a larger area for the sprites to be drawn onto.
    // We also are using this image's pixel coordinates as the coordinates
    // of our circle sprites.
    private BufferedImage drawing;

    public SimRenderer(SimData simData, BlockingQueue<BufferStrategy>
bufferStrategyQueue, int drawingWidth, int drawingHeight) {
        this.simData = simData;
        this.bufferStrategyQueue = bufferStrategyQueue;
        this.drawAreaBounds = new Rectangle(0, 0, drawingWidth, drawingHeight);

        // We draw almost all graphics to this image,
        // then stretch it over the
        // entire frame.
        // This allows a resize to make the sim bigger, as opposed to
        // just providing a larger area for the sprites to be drawn onto.
        drawing = GraphicsEnvironment.getLocalGraphicsEnvironment()
            .getDefaultScreenDevice().getDefaultConfiguration()
            .createCompatibleImage(drawingWidth, drawingHeight);
    }

    public synchronized void setComponentToDraw(Component componentToDraw)
    {
        this.componentToDraw = componentToDraw;
    }

    @Override

```

```

public void drawAreaChanged(int x, int y, int width, int height) {
    synchronized (drawAreaBounds) {
        drawAreaBounds.setBounds(x, y, width, height);
    }
}

public void renderLoop() {
// Wait for a buffer strategy from the queue, can't start the sim
// without being able to draw graphics
try {
    bufferStrategy = bufferStrategyQueue.poll(1, TimeUnit.SECONDS);
} catch (InterruptedException e) {
    // Thread should not be interrupted, this method is the lowest
    // method the thread should execute from this point on, as the
    // method has an infinite loop
    e.printStackTrace();
}
if (bufferStrategy == null) {
    System.err.println("BufferStrategy could not be made in render loop!");
    System.exit(1);
}
// For max accuracy, resetting the time since last update so
// animations and sprite positions remain in their standard first
// position
simData.resetTimeOfLastRender();
// Just loop and loop forever, update state and then draw.
while (true) {
    long nanoTimeAtStartOfRender = System.nanoTime();

    try {
        Graphics2D g = (Graphics2D) bufferStrategy.getDrawGraphics();
        drawSim(g);
        g.dispose();
        if (!bufferStrategy.contentsLost()) {
            bufferStrategy.show();
        }
    }
    // This catch is to allow the application to not stop
    // working when the application encounters the possible bug:
    // http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6933331
    // One work around to not encounter this is to Disable d3d
    // using -Dsun.java2d.d3d=false
    // Not sure why the bug is said to "... has no consequences
    // other than a stack trace dump in a console (no hang... ",
    // as people are generally not going to catch an
    // IllegalStateException...
}

```

```

// You can try to see if you can get the exception to print
// by resizing the window rapidly on the primary or secondary,
// or dragging the window off and on the primary monitor.
// This of course assumes you are using d3d
catch (IllegalStateException e) {
    e.printStackTrace();
    simData.logger.severe("Illegal state in render loop (" +
e.getStackTrace() + ")");
}

        waitUntilNextRender(nanoTimeAtStartOfRender);
    }
}

private synchronized void drawSim(Graphics2D g) {
    // Obtaining the graphics of our drawing image we use,
    // most of the graphics drawn are drawn to this object
    Graphics2D drawingBoard = drawing.createGraphics();
    simData.drawSimData(drawingBoard, drawing.getWidth(), drawing.getHeight());
    drawingBoard.dispose();
    final Graphics swingAndOtherGuiGraphics = g.create();
    synchronized (drawAreaBounds) {
        // The translate is needed to align our drawing of
        // components to their "clickable" areas (changes where 0, 0
        // actually is, comment it out and see what happens!)
        swingAndOtherGuiGraphics.translate(drawAreaBounds.x, drawAreaBounds.y);
        Graphics simGraphics = g.create();
        // Image call that scales and stretches the sim's graphics over
        // the entire frame
        // NOTE: This method of stretching graphics is not optimal.
        // This causes a computation of a stretched image each time. A
        // better implementation would be to cache an image of the
        // latest representation of a drawn sim,
        // and re-cache whenever there is a visible change (like color,
        // or its size, which would be due to a window resize), and
        // draw that cached image at the correct calculated location.
        // Additionally, it also causes the rendering to this image to
        // be done on the CPU. See the improvements section in the
        // tutorial.
        simGraphics.drawImage(drawing, drawAreaBounds.x,
            drawAreaBounds.y, drawAreaBounds.width,
            drawAreaBounds.height, null);
        // try {
            // File outputImage = new
File("Video/videoout"+imageRendered+".png");
            // ImageIO.write(drawing,"png",outputImage);

```

```

        // } catch (IOException e) {}
        // imageRendered++;
    simGraphics.dispose();
}
// componentToDraw is lazily set from the EDT during GUI creation or via main
menu dialog
if (componentToDraw != null) {
    // Paint our Swing components, to the graphics object of the
    // buffer, not the BufferedImage being used for the
    // application's sprites.
    // We do this, because Swing components don't resize on frame
    // resizes, they just reposition themselves, so we shouldn't
    // stretch their graphics at all.
try {
    SwingUtilities.invokeAndWait(new Runnable() {
        @Override
        public void run() {
            if (componentToDraw instanceof JComponent) {
                ((JComponent) componentToDraw).paintComponents
                    (swingAndOtherGuiGraphics);
            } else {
                componentToDraw.paintAll
                    (swingAndOtherGuiGraphics);
            }
        }
    });
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
} catch (InvocationTargetException e) {
    // should not happen
    e.printStackTrace();
}
}

// In addition, draw the FPS/UPS post stretch, so we always can read
// the info even if you shrink the frame really small.
// Grab the font height to make sure we don't draw the stats outside
// the panel, or over each other.

int fontHeight = g.getFontMetrics(g.getFont()).getHeight();
if(simData.getRunning()) {
    /// \todo draw text within transparent black box like the real time
plotter
    int curFPS = simData.getFPS();
    if(curFPS >= 24)
        swingAndOtherGuiGraphics.setColor(Color.green);
    else if((curFPS>=20)&&(curFPS<24))

```

```

        swingAndOtherGuiGraphics.setColor(Color.yellow);
    else
        swingAndOtherGuiGraphics.setColor(Color.red);
    swingAndOtherGuiGraphics.drawString("FPS: " + curFPS, 0,
fontHeight * 4);
    int curUPS = simData.getUPS();
    if(curUPS >= 10)
        swingAndOtherGuiGraphics.setColor(Color.green);
    else if((curUPS>=6)&&(curUPS<10))
        swingAndOtherGuiGraphics.setColor(Color.yellow);
    else
        swingAndOtherGuiGraphics.setColor(Color.red);
    swingAndOtherGuiGraphics.drawString("UPS: " + curUPS, 0,
fontHeight * 5);
    swingAndOtherGuiGraphics.setColor(Color.white);
    DecimalFormat myFormatter = new
DecimalFormat("###,###.##");
    swingAndOtherGuiGraphics.drawString("Time: " +
myFormatter.format(simData.getSimTime()), 0, fontHeight * 6);
}
else {
    swingAndOtherGuiGraphics.setColor(Color.white);
    swingAndOtherGuiGraphics.drawString("FPS: " +
simData.getFPS(), 0, fontHeight );
}
swingAndOtherGuiGraphics.dispose();
}

/**
 * Sleeps the current thread if there's still sometime the application
 * can wait for until the time the next update is needed.
 *
 * \param nanoTimeCurrentUpdateStartedOn Time that current update
 *          started
 */
private void waitUntilNextRender(long nanoTimeCurrentRenderStartedOn) {
    // Only sleep to maintain the update speed if speed is higher than
    // zero, because Thread.sleep(0) is not defined on what that
    // exactly does
    long currentRenderSpeed = simData.getCurrentWaitTimeBetweenRenders();
    if (currentRenderSpeed > 0) {
        // This could be more accurate by sleeping what's needed on
        // average for the past few seconds
        long timeToSleep = currentRenderSpeed - ((System.nanoTime() -
nanoTimeCurrentRenderStartedOn) / 10000000L);
        // If the speed of updating was so slow that it's time for

```

```

        // the next update, then choose 0
        timeToSleep = Math.max(timeToSleep, 0);
        // Again, avoiding Thread.sleep(0)
        if (timeToSleep > 0) {
            try {
                Thread.sleep(timeToSleep);
            } catch (InterruptedException e) {
                // It's okay if we're interrupted, program will just run
                // faster.
                Thread.currentThread().interrupt();
            }
        }
        // else
        // System.out.println("Render too slowly...LAG!");
    }
}
}

```

vdhils/Graphics/Speedometer.java:

```


/**
 *      \file Speedometer.java
 *
 *      \author Thomas Stevens
 *
 *      MainMenu contains GUI components to render splash screen's image
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE


```

```

*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****package vdhils.Graphics;
import vdhils.SimData;
//! Import Libraries
import java.awt.*;
import javax.swing.*;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.io.*;
/**  \class Speedometer
 *   \brief Speedometer class provides the dashboard which includes a speedometer
and the
 *
speedometer needle, a debug mileage 'LCD' readout, throttle and break
box meters
*
as well as all the functionality that intuitively goes with these dashboard
items.
*/
/// \bug Save/Load functionality doesn't load all data to objects (i.e. simData and
obstacleAdmin)
/// \bug Paused state disables simulation toolbar buttons
enum State {
    OFF, ON, BLINK
}
public class Speedometer extends Sprite {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
serialized output stream */
    private static final long serialVersionUID = 7989078461104748970L;

    private final int offset = 25;
    private Needle needle;
    private float mileage = 0.0f;
    public int x_pos;
    public int y_pos;
    private BoxMeter brakeBoxMeter;
    private BoxMeter throttleBoxMeter;
    private MileageMeter mileageMeter;
    private IndicatorLED rrtIndicator;
    private IndicatorLED collisionIndicator;
    public boolean mouseOverSpeedometer = false;
    private boolean slide = false;
}

```

```

    private String[] displayMsgs = { "RRT Disabled", "RRT Enabled", "...running",
        "...paused", "...error: ", "No Roadbounds" };

    public Speedometer(Dimension screen, String filepath, String needlefilepath) {
        super( screen, 0.5f, filepath, false ); // 0.5 == Scale TODO: add Scale
        param
            needle = new Needle( screen, needlefilepath );
            x_pos = (int)((screenSize.getWidth()/2)-((spriteWidth*spriteScale)/2));
            y_pos = (int)(screenSize.getHeight()-(spriteHeight-offset)*spriteScale);

            brakeBoxMeter = new BoxMeter("B",0,0.0f,6,44,Color.red);
            throttleBoxMeter = new BoxMeter("T",1,0.0f,6,44,Color.green);
            mileageMeter = new MileageMeter( x_pos+(spriteWidth/2)*spriteScale-
45f, y_pos+0.80f*spr  
iteHeight*spr  
iteScale, 90.0f, 20.0f, 2,"Courier New");
            rrtIndicator = new IndicatorLED( x_pos+0.65f*spr  
iteWidth*spr  
iteScale,
y_pos+0.85f*spr  
iteHeight*spr  
iteScale, 8f, 1, State.OFF );
            collisionIndicator = new IndicatorLED(
x_pos+0.325f*spr  
iteWidth*spr  
iteScale, y_pos+0.85f*spr  
iteHeight*spr  
iteScale, 8f, 0,
State.OFF );
    }
    public synchronized void writeSpeedometerToFile(String filepath) {
        try {
            // Serialize data object to a file
            ObjectOutputStream SpeedoOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
            SpeedoOut.writeObject(this);
            SpeedoOut.close();

            //Serialize data object to a byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            SpeedoOut = new ObjectOutputStream(bos);
            SpeedoOut.writeObject(this);
            SpeedoOut.close();
            byte[] buf = bos.toByteArray();
            SimData.logger.info("Save speedometer successfully");
        } catch (IOException e) {
            SimData.logger.warning("Failed to write speedometer object to
file (" + e.toString() + ")");
        }
    }
    public void setSlideIndicator(boolean sliding) {
        if(sliding)
            slide = true;
        else
            slide = false;
    }
}

```

```

        }
    protected void toggleRRTIndicator() {
        rrtIndicator.toggleState();
    }
    protected void setRRTIndicator(int RRTIndicator) {
        if(RRTIndicator==1)
            rrtIndicator.setState(State.ON);
        else
            rrtIndicator.setState(State.OFF);
    }
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        Composite c = g2d.getComposite();
        if(mouseOverSpeedometer)

            g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
0.3f));
            if(image.size() > 0)
                g2d.drawImage(
getImage(),x_pos,y_pos,(int)(spriteWidth*spriteScale),(int)(spriteHeight*spriteScale),
null);

            throttleBoxMeter.draw(g2d);
            brakeBoxMeter.draw(g2d);
            mileageMeter.draw(g2d);
            rrtIndicator.draw(g2d);
            collisionIndicator.draw(g2d);
            Color ogColor = g2d.getColor();
            if(slide) {
                g2d.setColor(Color.red);
                g2d.drawString("SLIDE", x_pos+(spriteWidth/2)*spriteScale-
g2d.getFontMetrics(g2d.getFont()).stringWidth("SLIDE")/2,
y_pos+0.55f*spriteHeight*spriteScale);
                g2d.setColor(ogColor);
            }
            needle.draw(g2d);

            g2d.setComposite(c);
        }
    public synchronized void update(float speed, float throttleMeter, float
brakeMeter, double deltaMileage, int RRTIndicator) {
        needle.update(speed);
        mileage += (float)deltaMileage;
        throttleBoxMeter.update(throttleMeter);
        brakeBoxMeter.update(brakeMeter);
        setRRTIndicator(RRTIndicator);
}

```

```

    }

private class Needle extends Sprite {
    private int needle_x_pos;
    private int needle_y_pos;
    private float needleAngle;
    private final float radPerMPH = 0.2425f/10.0f;
    private float zeroAngle = -79.25f*radPerMPH;
    public Needle(Dimension screen, String filepath) {
        super( screen, 0.5f, filepath, true); // 0.5 == Scale TODO: add
Scale param
        needle_x_pos = (int)((screenSize.getWidth()/2)-
((spriteWidth*spriteScale)/2));
        needle_y_pos = (int)(screenSize.getHeight()-(spriteHeight-
offset)*spriteScale);
        needleAngle = zeroAngle;
    }
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        AffineTransform affineTransform = new AffineTransform();
        affineTransform.translate( needle_x_pos,needle_y_pos );
        affineTransform.scale( spriteScale, spriteScale );
        affineTransform.rotate(needleAngle,
spriteWidth/2,0.66f*spriteHeight);
        if(image.size() > 0)
            g2d.drawImage( getImage(), affineTransform, null );
    }
    public synchronized void update(float speed) {
        needleAngle = zeroAngle + speed*radPerMPH;
    }
}
private class BoxMeter {
    private String meterName = "";
    private float meterValue;
    private float meter_xpos;
    private float meter_ypos;
    private int meterHeight;
    private int meterWidth;
    private Color meterColor;
    public BoxMeter(String meterName,int position,float initialValue,int
width, int height,Color meterColor) {
        this.meterName = meterName;
        if(position==0) {
            meter_xpos = (x_pos+(spriteWidth/2)*spriteScale)-
(spriteWidth/2)*spriteScale*0.725f;
            meter_ypos = y_pos+0.9f*spriteHeight*spr

```

```

    } else {
        meter_xpos =
(x_pos+(spriteWidth/2)*spriteScale)+(spriteWidth/2)*spriteScale*0.725f;
        meter_ypos = y_pos+0.9f*sprteHeight*sprteScale;
    }
    meterValue = initialValue;
    meterWidth = width;
    meterHeight = height;
    this.meterColor = meterColor;
}
//public BoxMeter() {}
protected void draw(Graphics g) {
    g.setColor(Color.black);
    g.fillRect((int)meter_xpos,(int)(meter_ypos-
(12+meterHeight)),meterWidth,meterHeight);
    g.setColor(meterColor);
    g.fillRect((int)(meter_xpos+2),(int)(meter_ypos-10-
meterHeight*(meterValue/100.0f)),meterWidth-4,(int)((meterHeight-
4)*(meterValue/100.0f)));
    g.drawString(meterName,(int)meter_xpos,(int)meter_ypos);
}
public synchronized void update(float newMeterValue) {
    meterValue = newMeterValue;
}
private class MileageMeter {
    private String[] lineText;
    private int numLines;
    private Color[] lineColor;    /// \todo Allow filling behind text of
different colors to indicate and highlight important information at different levels
    private RoundRectangle2D.Float mileageRect;
    private String fontName;

    public MileageMeter( float x, float y, float rectwidth, float rectheight, int
numLines, String fontName ) {
        mileageRect = new RoundRectangle2D.Float(x,
y,rectwidth,rectheight, 5, 5);
        this.fontName = fontName;
    }
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor( Color.gray );
        g2d.fill( mileageRect );
        g2d.setColor(Color.cyan);
        g2d.setFont( new Font(fontName, Font.BOLD, 8) );
    }
}

```

```

        g2d.drawString("Mileage: "+String.format("%.3f",mileage),
(float)mileageRect.getX()+2, (float)mileageRect.getY()+8);
        g2d.drawString("User info here", (float) mileageRect.getX()+2,
(float)mileageRect.getY()+18);
    }
}

private class IndicatorLED {
    private float LED_xpos;
    private float LED_ypos;
    private float LED_size;
    private int LED_color;
    private Color[] LED_colors = new Color[] { Color.green, Color.red,
Color.orange };
    private State state; // { off on blink}

    public IndicatorLED(float x, float y, float size, int ledcolor, State state) {
        LED_xpos = x;
        LED_ypos = y;
        LED_size = size;
        LED_color = ledcolor;
        this.state = state;
    }
    public void draw(Graphics g) {
        // if RRT enabled draw green indicator LED otherwise draw red
indicator LED
        Graphics2D g2 = (Graphics2D) g;

        g2.setColor(new Color(214,163,46));
        g2.fill(new Ellipse2D.Float(LED_xpos-1.5f, LED_ypos-1.5f,
LED_size+3f, LED_size+3f));
        if(state == State.ON)
            g2.setColor(LED_colors[LED_color]);
        else
            g2.setColor(Color.gray);
        g2.fill(new Ellipse2D.Float(LED_xpos, LED_ypos, LED_size,
LED_size));
    }
    protected void setColor(int newColor) {
        if(newColor < LED_colors.length)
            LED_color = newColor;
    }
    protected void setState(State state) {
        this.state = state;
    }
    protected void toggleState() {
        if(state != State.BLINK) {

```

```
        if(state == State.ON)
            state = State.OFF;
        else
            state = State.ON;
    }
}
```

## vdhils/Graphics/Sprite.java:

```
/***
*   \file Sprite.java
*   \brief Sprite class
*
*   Revisions:
*
*   License:
*   This file is copyright 2014 by T Stevens and released under the Lesser GNU
*   Public License, version 2. It intended for educational use only, but its use
*   is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
*****  
*****  
package vdhils.Graphics;  
import vdhils.SimData;
```

```

// Import Libraries
import java.awt.*;
import java.awt.image.*;
import javax.swing.ImageIcon;
import java.util.ArrayList;
import java.io.*;
import javax.swing.*;

/** \class Sprite
 *      \brief Sprite class
 */

public abstract class Sprite implements Serializable, Drawable {
    /** Serializable UID for saving/loadding functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
    serialized output stream */
    private static final long serialVersionUID = 7989078461104748971L;
    protected Dimension screenSize;      /**< Monitor's screen dimensions in pixels
    */
    public float    spriteScale;        /**< Sprite's scale for rendering to screen */
    public int      spriteWidth;       /**< Sprite's image width in pixels
    */
    public int      spriteHeight;      /**< Sprite's image height in pixels
    */
    /** List of sprite image filepaths - sprite may consist of multiple sub-sprites */
    protected ArrayList<String> imagestr = new ArrayList<String>();
    /** List of sprite image's loaded from image filepaths - sprite may consist of
    multiple sub-sprites */
    protected ArrayList<ImageIcon>    image = new ArrayList<ImageIcon>();
    protected boolean    visible;       /**< Boolean indicating if sprite is visible */
    protected float     depth = 1.0f;   /**< Depth value indicating sprite's alpha
    value (transparency) */
    protected String    filePath;      /**< String holding the filepath to
    the root sprite image */
    protected boolean    transparent;  /**< Boolean indicating if sprite's
    background should be made transparent */

    public abstract void draw(Graphics g);

    public Sprite( Dimension screen, float scale, String filePath, boolean transparent)
    {
        spriteScale = scale;

        addImage( filePath, transparent );

        screenSize = screen;
        visible = true;
        setVisible(true);
        this.filePath = filePath;
    }
}

```

```

        this.transparent = transparent;
    }
/*! No argument constructor for serializable operations (First non-serializable
class must have a no argument constructor for deserialization) */
public Sprite() {

}

protected void addImage(String imagePath,boolean transparent) {
    if( !(imagePath == null || imagePath.trim().equals("")) ) {
        imagestr.add( imagePath );
        java.net.URL imgURL = getClass().getResource(imagePath);
        BufferedImage mImage;
        try{
            mImage = javax.imageio.ImageIO.read(imgURL);
            int color = mImage.getRGB(0, 0);
            spriteWidth = mImage.getWidth();
            spriteHeight = mImage.getHeight();
            if(transparent == true)
                image.add( new
ImageIcon(makeColorTransparent(mImage, new Color(color))) );
            else
                image.add( new ImageIcon((Image) mImage));
        } catch(Exception e) {
            System.out.println("Pic error: "+e.toString() +
"+imagePath);
            SimData.logger.warning("Failed to load sprite with
imagePath ("+imagePath+) due to " + e.toString());
        }
    }
}
/***
 *      Function to return value indicating whether sprite is visible
 *      \return visible
 */
public boolean isVisible() {
    return visible;
}
/** Function to set value indicating whether sprite is visible
 *  \param visibility Indicates if sprite is visible
 */
protected void setVisible(boolean visibility) {
    visible = visibility;
}

/** Function to get sprite's scale

```

```

        *      \return spriteScale
        */
    public float getSpriteScale() {
        return spriteScale;
    }
    /**
     *
     */
    public AlphaComposite makeComposite() {
        int type = AlphaComposite.SRC_OVER;
        return(AlphaComposite.getInstance(type,depth));
    }
    /** Function to get misc. car textures stored at different indexes
     *      (i.e. wheel textures, skid/slide textures, etc...)
     *      /param index Index of misc. car texture to return
     *      \return Returns the misc. car sprite texture as an image
     */
    public Image getImage(int index) {
        return image.get(index).getImage();
    }
    /** Function to get car texture image stored at index 0 by default
     *      /return Returns car sprite texture as an image
     */
    public Image getImage() {
        return image.get(0).getImage();
    }
    /**
     *
     */
    public static Image makeColorTransparent(BufferedImage im,final Color color) {
        ImageFilter filter = new RGBImageFilter() {
            public int markerRGB = color.getRGB() | 0xFF000000;
            public final int filterRGB(int x,int y,int rgb) {
                if ((rgb | 0xFF000000) == markerRGB) {
                    // Mark the alpha bits as zero - transparent
                    return 0x00FFFFFF & rgb;
                } else {
                    // nothing to do
                    return rgb;
                }
            }
        };
        ImageProducer ip = new FilteredImageSource(im.getSource(),filter);
        return Toolkit.getDefaultToolkit().createImage(ip);
    }
}

```

## vdhils/Graphics/JFrameWithResizeListener.java:

```

/**
 *      \file JFrameWithResizeListener.java
 *      \brief JFrameWithResizeListener is a JFrame that tell's it's sole listener
 *              that it was created with the drawable bounds of it's frame on
resizes.
*
*      \author
*
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//****************************************************************************
*****  

package vdhils.Graphics;
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
/***
*      \class JFrameWithResizeListener
*      \brief JFrameWithResizeListener is a JFrame that tell's it's sole listener
*              that it was created with the drawable bounds of it's frame on
resizes.

```

```

*/
public class JFrameWithResizeListener extends JFrame implements ComponentListener {
    // Sole listener to the resizes of the frame
    private ResizeListener resizeListener;

    public JFrameWithResizeListener(ResizeListener resizeListener) {
        this.addComponentListener(this);
        this.resizeListener = resizeListener;
    }

    @Override
    public void componentResized(ComponentEvent e) {
        Insets insets = this.getInsets();
        resizeListener.drawAreaChanged(
            insets.left,
            insets.top,
            this.getWidth() - (insets.left + insets.right),
            this.getHeight() - (insets.top + insets.bottom));
    }

    @Override
    public void componentMoved(ComponentEvent e) {
        // Do nothing, not needed
    }
    @Override
    public void componentShown(ComponentEvent e) {
        // Do nothing, not needed
    }
    @Override
    public void componentHidden(ComponentEvent e) {
        // Do nothing, not needed
    }
}

```

vdhils/Graphics/GuiCreatorRunnable.java:

```

/**
 *      \file GuiCreatorRunnable.java
 *      \brief GuiCreatorRunnable is a runnable, that creates the GUI and a
BufferStrategy, and stores the reference to the strategy to later retrieve.
*
*      References:
*
*      Revisions:
*          \li 4/6/2014 TFS

```

```

*
*      License:
*  This file is copyright 2014 by T Stevens and released under the Lesser GNU
*  Public License, version 2. It intended for educational use only, but its use
*  is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
*  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
*  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
*  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
*  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
*  TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
*  OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
*  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
*  OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
*  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package vdhils.Graphics;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image(BufferedImage;
import java.lang.reflect.InvocationTargetException;
import java.util.Random;
import java.util.concurrent.*;
import java.awt.Toolkit;
/**  \class GuiCreatorRunnable
*   \brief GuiCreatorRunnable is a runnable, that creates the GUI and a
BufferStrategy, and stores the reference to the strategy to later retrieve.
```

```

*/
public class GuiCreatorRunnable implements Runnable {
    private int width;
    private int height;
    protected SimRenderer simRenderer;
    private JPanel panelContent;
    private BlockingQueue<BufferStrategy> bufferStrategyQueue;
    private JFrame frame;
    private String versNumber;
    /**
     * Constructs a GuiCreatorRunnableFuture, with the requested width
     * and height, and the SimRenderer object to update with a reference
     * to a SwingComponentDrawer, a SimData object to pass to the GUI controls,
     * and the BlockingQueue to store a BufferStrategy
     */
    public GuiCreatorRunnable(JFrame frame, int width, int height, SimRenderer
simRenderer, BlockingQueue<BufferStrategy> bufferStrategyQueue, JPanel
panelContent, String versNumber) {
        this.frame = frame;
        this.width = width;
        this.height = height;
        this.simRenderer = simRenderer;
        this.bufferStrategyQueue = bufferStrategyQueue;
        this.panelContent = panelContent;
        this.versNumber = versNumber;
    }
    @Override
    public void run() {
        frame.setTitle("VDHILS-" + versNumber);
        // Ignore repaints, as we will actively render the frame's graphics
        // ourselves
        frame.setIgnoreRepaint(true);
        // While we have the frame reference available, set it's content
        // pane to not be opaque.
        // The JFrame's content pane's background would otherwise paint over
        // any other graphics we painted ourselves
        if (frame.getContentPane() instanceof JComponent) {
            // JComponent's have a handy setOpaque method
            ((JComponent) frame.getContentPane()).setOpaque(false);
        } else {
            frame.getContentPane().setBackground(new Color(0, 0, 0));
        }
    }
}

```

```

    GraphicsEnvironment env =
GraphicsEnvironment.getLocalGraphicsEnvironment();
    GraphicsDevice device = env.getDefaultScreenDevice();
    boolean isFullScreen = device.isFullScreenSupported();
    // isFullScreen = false;
    frame.setUndecorated(isFullScreen);

    // frame.setUndecorated(true);
    frame.setResizable(!isFullScreen);
    if (isFullScreen) {
        // Full-screen mode
        device.setFullScreenWindow(frame);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.validate();
    } else {
        // Windowed mode
        frame.pack();
        frame.setVisible(true);
    }

    // Change width and height of window so that the available
    // screen space actually corresponds to what is passed, another
    // method is the Canvas object + pack()
    frame.setSize(width, height);
    Insets insets = frame.getInsets();
    int insetWide = insets.left + insets.right;
    int insetTall = insets.top + insets.bottom;
    frame.setSize(frame.getWidth() + insetWide, frame.getHeight() + insetTall);

    frame.add(panelContent);
    // Create the BufferStrategy, and store the reference to it
    frame.createBufferStrategy(2);
    try {
        bufferStrategyQueue.put(frame.getBufferStrategy());
    } catch (InterruptedException e) {
        // Should not be interrupted
        e.printStackTrace();
    }
    // SimUpdater will render all of the frame's components
    simRenderer.setComponentToDraw(frame.getContentPane());
}
}

```

vdhils/Graphics/Drawable.java:

```
package vdhils.Graphics;
```

```
import java.awt.Graphics;  
  
public interface Drawable {  
    public void draw(Graphics g);  
}
```

## Appendix D User Control Station (UCS) Code

UCS/Main.java

```
/***
 * \file Main.java
 *
 *
 *
 *
 *      An example of active rendering while double buffering
 *      The article on this source code can be found at:
 *
 *          http://jamesgames.org/resources/double_buffer/double_buffering_and_active_ren
dering.html
 *      Code demonstrates: - Active rendering in swing in a resizeable frame
 *                      - properly set width and height of a JFrame using Insets
 *                      - double buffering and active rendering via BufferStrategy
 *                      - usage of a high resolution timer for time based
 *                          animations
 *                      - stretching an application's graphics with resizes
 *                      - integrating Swing components with your active rendering
 *                          solution
 *
 *      Original Swing Active Rendering Code by:
 *          \author James Murphy
 *          \author Oracle
 *      Modified and Adapted by:
 *          \author Thomas Stevens
 */
* Copyright (c) 2011, Oracle and/or its affiliates. All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
*     - Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*
*     - Redistributions in binary form must reproduce the above copyright
*       notice, this list of conditions and the following disclaimer in the
*       documentation and/or other materials provided with the distribution.
*
*     - Neither the name of Oracle or the names of its
*       contributors may be used to endorse or promote products derived
*       from this software without specific prior written permission.
*/

```

```

/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****\todo Allow full-screen exclusive mode or windowed mode
package UCS;
// Import Libraries
import UCS.Graphics.*;
import UCS.GUI.*;
// GUI Imports
import javax.swing.*;
import javax.swing.plaf.LayerUI;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image BufferedImage;
import java.awt.Toolkit;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Random;
import java.util.concurrent.*;

```

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.beans.PropertyChangeEvent;
<太后
 * \class Main
 */
public class Main implements ActionListener {

    private static Main UCS;
    private final static Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    private static SimData simData;
    private static SimUpdater simUpdater;
    private static SimRenderer simRenderer;
    private BlockingQueue<BufferStrategy> bufferStrategyQueue;
    /// \todo Make width and height static
    private final int width;
    private final int height;
    private JFrame mainFrame;
    private MainMenu mainMenuPanel;
    private SettingsMenu settingsMenu;
    private HardwareInLoopSimulation sim;

    private JLayer<JPanel> jlayer;
    private WaitLayerUI layerUI;
    private final Timer stopper;

    private String versNumber;
    public static void main(String[] args) {
        // Constructor kicks off the GUI
        UCS = new Main((int)screenSize.getWidth()/2, (int)screenSize.getHeight()/2);
        // Start/run the simulation
        UCS.start();
    }

    class WaitLayerUI extends LayerUI<JPanel> implements ActionListener {
        private boolean mIsRunning;
        private boolean mIsFadingOut;
        private Timer mTimer;

        private int mAngle;
        private int mFadeCount;
        private int mFadeLimit = 15;
    }
}

```

```

@Override
public void paint (Graphics g, JComponent c) {
    int w = c.getWidth();
    int h = c.getHeight();

    // Paint the view.
    super.paint (g, c);

    if (!mIsRunning) {
        return;
    }

    Graphics2D g2 = (Graphics2D)g.create();

    float fade = (float)mFadeCount / (float)mFadeLimit;
    // Gray it out.
    Composite urComposite = g2.getComposite();
    g2.setComposite(AlphaComposite.getInstance(
        AlphaComposite.SRC_OVER, .5f * fade));
    g2.fillRect(0, 0, w, h);
    g2.setComposite(urComposite);

    // Paint the wait indicator.
    int s = Math.min(w, h) / 5;
    int cx = w / 2;
    int cy = h / 2;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setStroke(new BasicStroke(s / 4, BasicStroke.CAP_ROUND,
        BasicStroke.JOIN_ROUND));
    g2.setPaint(Color.white);
    g2.rotate(Math.PI * mAngle / 180, cx, cy);
    for (int i = 0; i < 12; i++) {
        float scale = (11.0f - (float)i) / 11.0f;
        g2.drawLine(cx + s, cy, cx + s * 2, cy);
        g2.rotate(-Math.PI / 6, cx, cy);
        g2.setComposite(AlphaComposite.getInstance(
            AlphaComposite.SRC_OVER, scale * fade));
    }

    g2.dispose();
}

public void actionPerformed(ActionEvent e) {
    if (mIsRunning) {
        firePropertyChange("tick", 0, 1);
        mAngle += 3;
    }
}

```

```

        if (mAngle >= 360) {
            mAngle = 0;
        }
        if (mIsFadingOut) {
            if (--mFadeCount == 0) {
                mIsRunning = false;
                mTimer.stop();
            }
        }
        else if (mFadeCount < mFadeLimit) {
            mFadeCount++;
        }
    }
}

public void start() {
    if (mIsRunning) {
        return;
    }

    // Run a thread for animation.
    mIsRunning = true;
    mIsFadingOut = false;
    mFadeCount = 0;
    int fps = 24;
    int tick = 1000 / fps;
    mTimer = new Timer(tick, this);
    mTimer.start();
}

public void stop() {
    mIsFadingOut = true;
}

@Override
public void applyPropertyChange(PropertyChangeEvent pce, JLayer l) {
    if ("tick".equals(pce.getPropertyName())) {
        l.repaint();
    }
}
/***
 * Constructor for ActiveRenderSkeleton
 *
 * \param width      The width of the program's inside portion of the frame
 * \param height     The height of the program's inside portion of the frame
 */

```

```

/*
public Main(final int width, final int height) {
    this.simData = new SimData(width, height);
        // Get a reference to the singleton.
        // SimData sdata = SimData.getInstance();
        // Protect singleton member variable from
        // multithreaded access.
        // synchronized(SingletonTest.class) {
            // if(simData == null) // If local reference is null...
                // simData = sdata; // ...set it to the singleton
        // }
    this.bufferStrategyQueue = new ArrayBlockingQueue<>(1);
    this.simRenderer = new SimRenderer(simData, bufferStrategyQueue,
width, height);
    this.simUpdater = new SimUpdater(simData);
        this.width = width;
        this.height = height;

    try {
        File versfile = new File("Info/VersionNumber.txt");
        FileReader fileReader = new FileReader(versfile);
        BufferedReader bufferedReader = new
BufferedReader(fileReader);
        String line = bufferedReader.readLine();
        if(line != null)
            versNumber = line;
        else
            versNumber = "";
    } catch(IOException e) {
        e.printStackTrace();
        simData.logger.severe("Unable to parse version number from
CompilerInfo");
    }

    mainFrame = new JFrameWithResizeListener(simRenderer);
    // This runnable will construct the GUI on the EDT, but also update our
    // SimUpdater with a SwingComponentDrawer object
    // that is created in this codebase, as well as store the created
    // BufferStrategy to use for active rendering to the BlockingQueue
    // passed.
    mainFrame.setAlwaysOnTop(true);
    mainMenuPanel = new MainMenu(this, width, height, versNumber);
    settingsMenu = new SettingsMenu(this, width, height, this.simData);

    Runnable guiCreator = new GuiCreatorRunnable(mainFrame, width, height,
simRenderer, bufferStrategyQueue, mainMenuPanel, versNumber);

```

```

try {
    // Invoke swing code on EDT thread (Swing code always executes
    on EDT - this method ensures it)
    SwingUtilities.invokeAndWait(guiCreator);
} catch (InterruptedException | InvocationTargetException e) {
    e.printStackTrace();
    simData.logger.severe("Failed to create GUI on EDT (" +
e.getStackTrace() + ")");
}
layerUI = new WaitLayerUI();
stopper = new Timer(1000, new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        layerUI.stop();
        simData.inputEnabled = true;
    }
});
stopper.setRepeats(false);
}

/**
 * Starts the updating of the sim's state such as animation and movement
 * in other thread(s).
 */
void start() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            simRenderer.renderLoop();
        }
    }).start();
}

public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();
    System.out.println(action);
    simData.logger.info("MainListener action: " + action);
    if(simData.inputEnabled) {
        simData.inputEnabled = false;
        if(action.equals("Start")) {

simData.initializeEmbeddedHardware(settingsMenu.getSerialPort());
sim = new
HardwareInLoopSimulation(simUpdater,simData,mainFrame,width,height);
simData.setSimulation(sim);

if(jlayer != null)

```

```

        mainFrame.remove(jlayer);
    else
        mainFrame.remove(mainMenuPanel);
    jlayer = new JLayer<JPanel>(sim, layerUI);

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }

        simData.setRunning(true);
    }
if(action.equals("Settings")) {
    if(jlayer != null)
        mainFrame.remove(jlayer);
    else
        mainFrame.remove(mainMenuPanel);
    jlayer = new JLayer<JPanel>(settingsMenu, layerUI);

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();

        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
if(action.equals("Back")) {
    /// \todo Prompt user: Save data before returning to main
menu?
    /// \todo Save logged data (and log data) functionality

    if(jlayer != null)
        mainFrame.remove(jlayer);
    else
        mainFrame.remove(sim);
    jlayer = new JLayer<JPanel>(mainMenuPanel, layerUI);
        simData.setRunning(false);
    simData.setPaused(true);
    /// \todo sim.clearData( time and simulation objects and
parameters )
    simData.setSimTime(0.0);
}

```

```

        try {
            sim.closeJoystick();
        } catch(Exception Nocontroller) {
            simData.logger.info("No Joystick to Close");
        }

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
    if(action.equals("BackFromSettings")) {
        if(jlayer != null)
            mainFrame.remove(jlayer);
        jlayer = new JLayer<JPanel>(mainMenuPanel, layerUI);

        mainFrame.add (jlayer);
        mainFrame.revalidate();
        mainFrame.repaint();
        layerUI.start();
        if (!stopper.isRunning()) {
            stopper.start();
        }
    }
    if(action.equals("Quit")) {
        // todo Replace try statements with if's to perform
        operation only if a joystick or BB are connected
        try {
            sim.close();
        } catch(Exception exp) {
            simData.logger.info("Error
while closing simulation "+("+"+exp.getStackTrace()+""));
        }

        mainFrame.setVisible(false);
        mainFrame.dispose();

        System.exit(0);
    }
}
}

```

UCS/HardwareInLoopSimulation.java

/\*\*

```

*      \file HardwareInLoopSimulation.java
*      \brief Hardware in loop Simulation Utilizing a EulerIntegration scheme.
*
*
*
*      References:
*
*
*      Revisions:
*          \li 2/6/2013 TFS
*          \li 11/19/2013 TFS
*          \li 4/23/2014 TFS
*          \li 8/15/2014 TFS
*
*
*      License:
*      This file is copyright 2013 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
*****  

*****  

/// \todo Implement rewind, play, pause, feature for simulation
/// \todo Makefile to erase saved simulations and log files
/// \todo Borders

package UCS;
// Import Libraries
import UCS.GUI.*;
```

```

import UCS.Graphics.*;
// GUI Imports
import javax.swing.*;
import javax.swing.JDialog;
import javax.swing.BorderFactory;
import javax.swing.border.Border;
import javax.swing.filechooser.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.ArrayList;
import java.util.*;
import java.text.*;
// Joystick API Imports
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;
// File Operation Imports
import java.io.File;
import static java.nio.file.StandardOpenOption.*;
import java.nio.file.Files;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
// Error Handling Imports
import java.io.IOException;
// Serializable Import
import java.io.*;
import java.awt.Toolkit;

/**
 * \class HardwareInLoopSimulation
 * \brief Hardware in loop Simulation utilizes a simple Euler numerical integration
scheme to model car dynamics in a real-time simulated environment. The simulated
environments consists of
 *          other AI drivers and their vehicles & behaviors (traffic), roadways,
and road cones spaced evenly along the each side of each roadway. The user vehicle is
controlled through a usb
 *          gamecontroller featuring a force-feedback steering wheel and
throttle and braking floor pedals. In addition, the user vehicle simulates a LiDAR sensor
with adjustable range, resolution,

```

```

*           and scan frequency in order to identify threatening obstacles within
the simulated environment and steer the vehicle according to path-planning algorithms
both of which run on an embedded
*                   development board (such as the BeagleBone or raspberry pi) . The
vehicle is mounted on a dynamometer.
*
*           This class is responsible for instantiating all the roadways, obstacles, and
environment as well as user controls and data streams & communications to kick off the
simulation. Simulation starts
*           in a PAUSED state, which may be toggled by pressing the space bar. This
class also serves as the base JPanel with a reference to its parent JFrame and
ActionListener to handle GUI manipulation.
*
*/
public class HardwareInLoopSimulation extends JPanel implements
ControllerEventListener, FeatureNotSupportedEventListener, Serializable, Drawable {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to the
serialized output stream */
    private static final long serialVersionUID = 7989078461104748962L;
    /** Specifies if this is the first time and indicates whether to create the simulation
update thread */
    private boolean firstEnable = true;
    /** Specifies the "PAUSED" font used during rendering */
    private Font pausedFont = new Font("Arail", Font.BOLD, 36);
    /** Update thread responsible for proper simulation timing and updating of
simulated vehicle dynamics and environment in real-time */
    private Thread updateThread;
    /** List of possible FFJoysticks */           /// \todo Clear this arraylist once
joystick is set
    private transient ArrayList<FFJoystick> joysticks;
    /** Specifies the user's FFJoystick that is used to drive the user's vehicle */
    private transient static FFJoystick joystick;
    /** Specifies the user's FFJoystick effect to realize the aligning moment at the
steering wheel caused by the car dynamics */
    private static Effect eff;
    /** Filechooser allowing user to specify a previously saved simulation in order to
load it */
    private final JFileChooser fc;
    /** Indicates whether the measure tool is active (via GUI) */
    private boolean measureToolEnable = false;
    /** Point's specified by user to be analyzed by the measure tool */
    private ArrayList<Point> clickPoints = new ArrayList<Point>();

    private static SimData simData;
    private final SimUpdater simUpdater;

```

```

private Speedometer speedometer;

/** Button to exit application */
private JButton exitButton;
/** Holds reference to the main JFrame container */
private static JFrame container;
/** Button to switch back to main menu saving all logged data to allow for post-
processing */
private JButton backButton;
private JToolBar toolbar;
private JButton toolBarExit;
private JButton toolBarHelp;
private int width;
private int height;

private Image cursorimage;
private Toolkit toolkit;

private byte DELIMITER = ' ';

/**
 * Construct our sim and set it running.
 */
public HardwareInLoopSimulation(SimUpdater simUpdater, SimData simData,
final JFrame container, final int width, final int height) {
    // Get references to simulation data and update objects
    this.simUpdater = simUpdater;
    this.simData = simData;
    this.width = width;
    this.height = height;
    this.container = container;

    /// \todo Different customizable dashboard: speedometer, RRT indicator
LED,
    speedometer = new Speedometer( new Dimension(width,height),
"/Resources/Dashboard.png", "/Resources/spedneedle2.png" );
    // JPanel panel = (JPanel) container.getContentPane();
    setFocusable(true);
    addKeyBindings(this);
    // Initialize joystick
    initializeJoystick();
    // Create a file chooser
    fc = new JFileChooser();
    fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    // Setting up the swing components
    toolbar = new JToolBar();
}

```

```

        toolbar.setFloatable(false);
        // Create toolbar icon's for each toolbar button
        ImageIcon iconExit = new ImageIcon("./Resources/Icons/ExitIcon.png");
        ImageIcon iconHelp = new
        ImageIcon("./Resources/Icons/HelpIcon.png");

        toolBarExit = new JButton(iconExit);
        toolBarHelp = new JButton(iconHelp);

        toolBarExit.setFocusable(false);
        toolBarHelp.setFocusable(false);

        toolBarExit.setToolTipText("Click this button to exit application");
        toolBarHelp.setToolTipText("Click this button to view the README and
project description file");

        toolbar.add(toolBarHelp);
        toolbar.addSeparator(new Dimension((int)(width - 3*32),32));
        toolbar.add(toolBarExit);

        toolBarExit.setActionCommand("Quit");
        toolBarExit.setMnemonic('Q');

        GridLayout holderLayout = new GridLayout(1, 0);
        holderLayout.setHgap(500);

        this.setLayout(new BorderLayout());
        this.add(toolbar, BorderLayout.NORTH);
        this.setOpaque(false);
    }

    public void close() {
        try {
            closeJoystick();
            simData.closeJoystick();
        } catch(Exception Nocontroller) {    simData.logger.info("No Joystick to
Close");}
        simData.closeEmbeddedHardware();
    }

    public void initializeJoystick() {
        simData.logger.info("Initializing JoystickManager...");
        System.out.println("Initializing JoystickManager...");
        try {
            JoystickManager.init();
            joysticks = JoystickManager.getAllFFJoysticks();

```

```

        joystick = JoystickManager.getFFJoystick();
    }
    catch (FFJoystickException e) {
        e.printStackTrace();
        simData.logger.warning("Joystick Initialization Error: " +
e.getMessage());
    }
    if (joysticks.isEmpty()) {
        System.out.println("No Joysticks Detected"); // TODO No
joysticks detected resort to keyed input and enable user-input via keyboard
        simData.logger.info("No Joysticks Detected");
        simData.setJoystickControl(false);
    }
    else {
        simData.setJoystickControl(true);
        ControllerEventManager.addControllerEventListener(this);

        FeatureNotSupportedEventManager.addFeatureNotSupportedEventListener(this);

        setJoystick(joystick);
    }
}

public void addKeyBindings( JComponent jc ) {

    jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_SPACE, 0, false), "Space pressed");
    jc.getActionMap().put("Space pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            System.out.println("Space pressed");
            // if(simData.inputEnabled || !simData.getRunning()) {
            if(simData.getPaused()) {
                simData.setPaused(false);
                if(firstEnable) {
                    updateThread = new Thread(new
Runnable() {
            @Override
            public void run() {
                simUpdater.updateLoop();
            }
        });
    }
    updateThread.start();
}

```

```

        }
        else {
            simData.setPaused(true);
            updateThread.stop();
            // updateThread.interrupt();
        }
    }
});

```

```
jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.
```

```
getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
```

```
jc.getActionMap().put("Escape pressed", new AbstractAction() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent ae) {
```

```
        if(simData.getJoystickControl()) {
```

```
            closeJoystick();
```

```
            try {
```

```
                simData.closeJoystick();
```

```
            } catch(Exception nullJoystick) {
```

```
simData.logger.warning("No Joystick to Close"); }
```

```
}
```

```
        simData.closeEmbeddedHardware();
```

```
        System.exit(0);
```

```
}
```

```
});
```

```
jc.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.
```

```
getKeyStroke(KeyEvent.VK_SPACE, 0, true), "Space released");
```

```
jc.getActionMap().put("Space released", new AbstractAction() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent ae) {
```

```
        System.out.println("Space released");
```

```
}
```

```
});
```

```
}
```

```
@Override
```

```
public void controllerEventOccured(AdvancedControllerEvent event) {
```

```
    // we simply output it. but there are different classes of Events -
```

```
    // we could also distinguish between them and do more advanced stuff.
```

```
    //      System.out.println(event);
```

```
}
```

```

@Override
public void featureNotSupportedEventOccured(FeatureNotSupportedEvent event)
{
    System.out.println("Joystick feature not supported ("+event+ ")");
    simData.logger.warning("Joystick feature not supported +" + event);
}
public void setJoystick(FFJoystick js) {
    joystick = js;
    if(joystick.isFFJoystick()) {
        simData.logger.info("Joystick set to: " + joystick.getName());
        System.out.println("Joystick set to: " + joystick.getName());
        if(joystick.getName().equals("Logitech Force 3D Pro
USB")||joystick.getName().equals("Logitech Driving Force GT USB")) {
            // we set different dead zones for different axes
            // \todo allow user to specify dead zone
            // joystick.setDeadZone(0, 0.01f);
            // joystick.setDeadZone(1, 0.01f);
            // joystick.setDeadZone(2, 0.2f);
            // joystick.setDeadZone(3, 0.2f);
        }
        simData.setJoystickFF(true);
    }
    else {
        simData.logger.warning(" -> ForceFeedback not supported");
        System.out.println(" -> ForceFeedback not supported");
        simData.setJoystickFF(false);
    }
    simData.setJoystick(joystick);
}
public synchronized void closeJoystick() {
    simData.closeJoystick();
    JoystickManager.close();
}
public void draw(Graphics g) {
    speedometer.draw(g);
    if(simData.getPaused()) {
        g.setColor(Color.red);
        g.setFont(pausedFont);
        g.drawString("PAUSED", width/2-
g.getFontMetrics().stringWidth("PAUSED")/2, height/2);
    }
}
public void update( float speed, float acceleration, float brakes, float mileage) {
    speedometer.update( speed, acceleration, brakes, mileage,
(int)simData.getThreat(), (int)simData.getValidPath() );
}

```

```
}
```

## UCS/SerialCommunicationJSSC.java

```
/***
 *      \file SerialCommunication.java
 *      \brief SerialCommunication is a class that performs serial tasks in order to
 * communicate with the BeagleBone embedded device by managing setup, listening, and
 * writing data to the serial port.
 *
 *      References:
 *
 *      Revisions:
 *          \li 4/18/2014 TFS
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```
package UCS;
// Import java API's/libraries
import java.io.InputStream;
```

```

import java.io.*;
import jssc.SerialPort;
import jssc.SerialPortList;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.*;
import java.nio.ByteBuffer;
/**  \class SerialCommunication
 *   \brief SerialCommunication is a class that performs serial tasks in order to
communicate with the BeagleBone embedded device by managing setup, listening, and
writing data to the serial port.
*/
public class SerialCommunicationJSSC implements Runnable, Serializable,
SerialPortEventListener {
    /** Serializable UID for saving/loading functionality */
    private static final long serialVersionUID = 7989078461104748972L;
    private SerialPort serialPort;
    /** Buffer for steerangle sent to embedded hardware for processing */
    private volatile byte steerAngle; // outgoing data - accessed by multiple threads
need to be volatile
    /** Buffer for throttle sent to embedded hardware for processing */
    private volatile byte throttle; // outgoing data - accessed by multiple threads need
to be volatile
    /** Measured speed return from embedded hardware utilizing wheel encoders
(average of 4 wheel encoders is returned) */
    private volatile float speed; // incoming data - accessed by multiple threads need
to be volatile

    private volatile byte threat;
    private volatile byte goodPath;

    private String PORT_NAMES[] = {
        /// \todo Allow setting of embedded serial port from combo box on menu
        "COM17",
        "COM6",
        "COM4"
    };
    private String PORT_NAME = "";
    private InputStream input;
    private OutputStream output;
    private int TIME_OUT = 3000;
}

```

```

private int DATA_RATE = 115200;

    //some ascii values for for certain things
public final static byte SPACE_ASCII = 32;
public final static byte DASH_ASCII = 45;
public final static byte NEW_LINE_ASCII = 10;

    public SerialCommunicationJSSC(String searchPorts[],int dataRate,int timeOut) {
        this(searchPorts, dataRate);
        TIME_OUT = timeOut;
    }
    public SerialCommunicationJSSC(String searchPorts[],int dataRate) {
        this(searchPorts);
        DATA_RATE = dataRate;
    }
    public SerialCommunicationJSSC(String searchPorts[]) {
        PORT_NAMES = searchPorts;
    }
    public SerialCommunicationJSSC(String comPort) {
        PORT_NAME = comPort;
    }
    public SerialCommunicationJSSC() {

    }
    public float getMeasuredSpeed() {
        return speed;
    }
    public byte getThreat() {
        return threat;
    }
    public byte getValidPath() {
        return goodPath;
    }
    private void addSerialPortEventListener() {
        try {
            int mask = SerialPort.MASK_RXCHAR;// +
SerialPort.MASK_CTS + SerialPort.MASK_DSR;
            serialPort.setEventsMask(mask);
            serialPort.addEventListerner(this);
        } catch(SerialPortException ex) {
            System.err.println(ex.toString());
            SimData.logger.warning("Serial connection to embedded hardware
failed: "+ex.toString());
        }
    }
}

```

```

public void setSteerAngle(byte steerAngle) {
    this.steerAngle = steerAngle;
}

public void setThrottle(byte throttle) {
    this.throttle = throttle;
}

public static byte[] toByteArray(double value) {
    byte[] bytes = new byte[8];
    ByteBuffer.wrap(bytes).putDouble(value);
    return bytes;
}

public static byte[] toByteArray(float value) {
    byte[] bytes = new byte[4];
    ByteBuffer.wrap(bytes).putFloat(value);
    return bytes;
}

public boolean connect() {
    System.out.println("Connecting...");
    SimData.logger.info("Connecting to embedded hardware serial port at:
"+this.getClass().getName());

    if((PORT_NAME != null) && (PORT_NAME != ""))
        serialPort = new SerialPort(PORT_NAME);
    else
        serialPort = new SerialPort(PORT_NAMES[0]);

    try{
        serialPort.openPort();//Open serial port
        serialPort.setParams(DATA_RATE,
        SerialPort.DATABITS_8,
        SerialPort.STOPBITS_1,
        SerialPort.PARITY_NONE);//Set params. Also you can set params
by this string: serialPort.setParams(9600, 8, 1, 0);

        System.out.println("Connected ");
        SimData.logger.info("Connected and awaiting serial input on port:
"+this.getClass().getName());
        addSerialPortEventListener();
        return true;
    } catch (Exception e) {
        System.err.println(e.toString());
        SimData.logger.warning("Serial connection to embedded hardware
failed: "+e.toString());
        return false;
    }
}

```

```

    }
    public void disconnect() {
        System.out.println("Disconnecting...");

        try {
            serialPort.closePort(); //Close serial port
            System.out.println("System Status: Closing serial connection");
        }
        catch (Exception exe) {
            SimData.logger.severe("Disconnect from Embedded Hardware Failed!
(" + exe.toString() + ")");
        }
    }
    public float getSpeed() {
        return speed;
    }
    public void run() {
        // System.out.println("RUN");
        try {
            // System.out.println("Out " + steerAngle + " " + throttle);
            // serialPort.writeByte(steerAngle);
            // serialPort.writeByte(throttle);
            byte[] outputBytes = new byte[2];
            outputBytes[0] = steerAngle;
            outputBytes[1] = throttle;
            serialPort.writeBytes(outputBytes);
        } catch (Exception exp) { System.out.println("Serial port
exception"); }
    }
    public static String[] getSerialPorts() {
        return SerialPortList.getPortNames();
    }
    public void serialEvent(SerialPortEvent event) {
        try {
            byte buffer[] = serialPort.readBytes(2);
            // Two first bytes is the measured speed from the embedded
            hardware times a factor of 100 (limit to 2 decimal places for speed!)
            // MSB of upper byte indicates if threat is enabled
            int result = buffer[0] & 0xff | ((buffer[1] << 8) & 0x7F);
            speed = result / 100.0f;
            threat = (byte)((buffer[1] & 0x80) >> 7);
            goodPath = (byte)((buffer[1] & 0x40) >> 6);

        } catch (SerialPortException ex) {
            SimData.logger.severe("Failed to read serial event from embedded
hardware");
        }
    }
}

```

```
        }
    }
}
```

UCS/SettingsData.java

```
/*
 *      \file SettingsData.java
 *      \brief Settings Data class
 *
 *      Revisions:
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****  
*****
```

package UCS;

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event;
```

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
import java.lang.reflect.InvocationTargetException;
import java.util.Random;
import java.util.concurrent.*;
import java.io.*;
import java.util.Hashtable;
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;
/**  \class SettingsData
 *   \brief Settings Data class
 */
public class SettingsData implements Serializable {
    /** Serializable UID for saving/loading functionality */
    private static final long serialVersionUID = 7989078461104748974L;
    private GraphicsSettings graphicsSettings;
    private ControllerSettings controllerSettings;
    private SimulationSettings simulationSettings;
    private EmbeddedSettings embeddedSettings;

    public SettingsData() {
        graphicsSettings = new GraphicsSettings();
        controllerSettings = new ControllerSettings();
        simulationSettings = new SimulationSettings();
        embeddedSettings = new EmbeddedSettings();
    }
    public GraphicsSettings getGraphicsSettings() {
        return graphicsSettings;
    }
    public ControllerSettings getJoystickSettings() {
        return controllerSettings;
    }
    public SimulationSettings getSimulationSettings() {

```

```

        return simulationSettings;
    }
    public EmbeddedSettings getEmbeddedSettings() {
        return embeddedSettings;
    }
    protected void setJoystick( FFJoystick joystick ) {

    }
    private class GraphicsSettings extends JPanel implements ItemListener,
    ActionListener {
        private Color longVelSignalColor;
        private Color latVelSignalColor;
        private Color speedSignalColor;
        private Color steerAngleSignalColor;
        private Color angVelSignalColor;
        private Color trailColor;
        private Color obstacleTrailColor;

        private final Color[] colors = { Color.black, Color.blue,
Color.cyan, Color.darkGray,
                                         Color.gray, Color.green, Color.lightGray,
Color.magenta,
                                         Color.orange, Color.pink, Color.red, Color.white,
Color.yellow };

        private JCheckBox plotSpeedometer;
        private JCheckBox plotTrail;
        private JCheckBox plotRealTimeData;
        private final JComboBox longVelocityColor = new
JComboBox(colors);
        private final JComboBox latVelColor = new JComboBox(colors);

        private JCheckBox plotLidarScan;
        private JCheckBox plotSegmentedObstacles;
        private JCheckBox plotThreateningObstacles;
        private JCheckBox plotCarTexture;
        private JCheckBox plotCarWheelTexture;

        private boolean renderCarTextures;
        private boolean renderWheelTextures;
        private boolean renderSpeedometer;

        private boolean realTimePlotterEnabled;
        private boolean realTimeDataLegend;
        private boolean renderLidarScan;
    }
}

```

```

private boolean renderObstacles;
private boolean renderThreateningObstacles;
private boolean renderTrail;
private boolean obstacleTrailEnabled;

private double realTimePlotterScale;
private double speedometerScale;
private double trailSize;
private double obstacleTrailSize;

private int trailLength;
private int obstacleTrailLength;
private int carGraphic;

public GraphicsSettings() {

    setBackground(SimData.CalPoly_CREAM);

    renderSpeedometer = true;
    realTimePlotterEnabled = true;
    renderLidarScan = true;
    renderObstacles = false;
    renderThreateningObstacles = false;
    renderTrail = true;
    renderCarTextures = true;
    renderWheelTextures = true;

    longVelocityColor.setMaximumRowCount(5);
    // longVelocityColor.setEditable(true);
    longVelocityColor.setRenderer(new
        ColorComboBoxEditor());
    longVelocityColor.setOpaque(true);
    // longVelocityColor.setLightWeightPopupEnabled(false);
    longVelocityColor.setFocusable(true);
    longVelocityColor.setVisible(true);
    longVelocityColor.addActionListener(this);

    plotSpeedometer = new JCheckBox("Plot Speedometer");

    plotSpeedometer.setBackground(SimData.CalPoly_CREAM);
    plotSpeedometer.setSelected( renderSpeedometer );

    plotTrail = new JCheckBox("Plot Trail");
    plotTrail.setBackground(SimData.CalPoly_CREAM);
    plotTrail.setSelected( renderTrail );
}

```

```

        plotRealTimeData = new JCheckBox("Plot Real Time
Data");

        plotRealTimeData.setBackground(SimData.CalPoly_CREAM);
        plotRealTimeData.setSelected( realTimePlotterEnabled );

        plotLidarScan = new JCheckBox("Plot Lidar Scan Data");
        plotLidarScan.setBackground(SimData.CalPoly_CREAM);
        plotLidarScan.setSelected( renderLidarScan );

        plotSegmentedObstacles = new JCheckBox("Plot
Segmented Obstacles from Lidar data");

        plotSegmentedObstacles.setBackground(SimData.CalPoly_CREAM);
        plotSegmentedObstacles.setSelected( renderObstacles );

        plotSegmentedObstacles.setEnabled(false);

        plotThreateningObstacles = new JCheckBox("Plot
Threatening Obstacles from Lidar data");

        plotThreateningObstacles.setBackground(SimData.CalPoly_CREAM);
        plotThreateningObstacles.setSelected(
renderThreateningObstacles );

        plotThreateningObstacles.setEnabled(false);

        plotCarTexture = new JCheckBox("Plot Car Texture
(setting to false renders a wireframe model)");

        plotCarTexture.setBackground(SimData.CalPoly_CREAM);
        plotCarTexture.setSelected( renderCarTextures );

        plotCarWheelTexture = new JCheckBox("Plot Tire Texture
(setting to false renders a wireframe model)");

        plotCarWheelTexture.setBackground(SimData.CalPoly_CREAM);
        plotCarWheelTexture.setSelected( renderWheelTextures );

        plotSpeedometer.addItemListener( this );
        plotTrail.addItemListener( this );
        plotRealTimeData.addItemListener( this );
        plotLidarScan.addItemListener( this );
        plotSegmentedObstacles.addItemListener( this );
        plotThreateningObstacles.addItemListener( this );
        plotCarTexture.addItemListener( this );

```

```

        plotCarWheelTexture.addItemListener( this );

        /// \todo Set components not visible depending on selected
checked items

        JPanel checkPanel = new JPanel( new GridLayout(0,1) );
        checkPanel.setBackground(SimData.CalPoly_CREAM);
        checkPanel.add( new JLabel("Render Settings:") );
        checkPanel.add( plotSpeedometer );
        checkPanel.add( plotTrail );
        checkPanel.add( plotRealTimeData );
        checkPanel.add( longVelocityColor );
        checkPanel.add( plotLidarScan );
        checkPanel.add( plotSegmentedObstacles );
        checkPanel.add( plotThreateningObstacles );
        checkPanel.add( plotCarTexture );
        checkPanel.add( plotCarWheelTexture );
        checkPanel.setOpaque(true);
        add( checkPanel );
    }

    /** Listens to the check boxes. */
    @Override
    public void itemStateChanged(ItemEvent e) {
        Object source = e.getItemSelectable();

        if (source == plotSpeedometer) {
            renderSpeedometer = plotSpeedometer.isSelected();
            System.out.println("speedometer select");
        } else if (source == plotTrail) {
            System.out.println("trail select");
        } else if (source == plotRealTimeData) {
            System.out.println("real time data select");
        }

        //Now that we know which button was pushed, find out
        //whether it was selected or deselected.
        if (e.getStateChange() == ItemEvent.DESELECTED) {

        }

        //Apply the change to the string.
        //choices.setCharAt(index, c);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();

```

```

        Color selectedName = (Color)cb.getSelectedItem();
        System.out.println("Event: "+e+" Source: "+cb);
        cb.setBackground(selectedName);
    }
}

private class ControllerSettings extends JPanel implements
ChangeListener {

    private boolean controllerEnabled;
    private double sensitivity;
    private double deadzone;
    private double steeringGearReduction;
    private boolean forceFeedbackEnabled;
    private float XDZ, YDZ, ZDZ;

    static final int SENS_MIN = 0;
    static final int SENS_MAX = 10;
    static final int SENS_INIT = 5; //initial frames per second

    private JSlider joystickSensitivity;
    private JTextField XDZtext, YDZtext, ZDZtext;
    private JComboBox<String> comPortSelect;

    public ControllerSettings() {
        setBackground(SimData.CalPoly_CREAM);

        joystickSensitivity = new JSlider(JSlider.HORIZONTAL,
SENS_MIN, SENS_MAX, SENS_INIT);
        joystickSensitivity.addChangeListener( this );
        joystickSensitivity.setMajorTickSpacing( 1 );
        joystickSensitivity.setPaintTicks(true);
        //Create the label table.
        Hashtable<Integer, JLabel> labelTable = new
Hashtable<Integer, JLabel>();
        //PENDING: could use images, but we don't have any good
ones.
        labelTable.put(new Integer( 0 ), new JLabel("Stop") );
        //new JLabel(createImageIcon("images/stop.gif") );
        labelTable.put(new Integer( SENS_MIN/10 ), new
JLabel("Sensitive") );
        //new JLabel(createImageIcon("images/slow.gif") );
        labelTable.put(new Integer( SENS_MAX ), new
JLabel("Not Sensitive") );
        //new JLabel(createImageIcon("images/fast.gif") );
        joystickSensitivity.setLabelTable(labelTable);
    }
}

```

```

XDZtext = new JTextField();
XDZtext.setText("0.0");
YDZtext = new JTextField();
YDZtext.setText("0.0");
ZDZtext = new JTextField();
ZDZtext.setText("0.0");

JPanel checkPanel = new JPanel( new GridLayout(0,1) );
checkPanel.setBackground(SimData.CalPoly_CREAM);
checkPanel.add( new JLabel("Joystick Settings") );
checkPanel.add( new JLabel("Steering Sensitivity") );
checkPanel.add( joystickSensitivity );
checkPanel.add( new JLabel("Set Throttle (X-Axis) dead
zone"));
checkPanel.add( XDZtext );
checkPanel.add( new JLabel("Set Brake (Y-Axis) dead
zone"));
checkPanel.add(YDZtext );
checkPanel.add( new JLabel("Set Steer (Z-Axis) dead
zone"));
checkPanel.add(ZDZtext );
checkPanel.add( new JLabel("Keyboard Settings") );

add(checkPanel);
}

/** Listen to the slider. */
@Override
public void stateChanged(ChangeEvent e) {
    JSlider source = (JSlider)e.getSource();
    if (!source.getValueIsAdjusting()) {
        int sensitivity = (int)source.getValue();
        if (sensitivity == 0) {
            ;
        } else {
            ;
        }
    }
}

private class SimulationSettings extends JPanel implements
ActionListener {
    private int FPSlimit;
    private boolean limitFPS;
    private String recordFilePath;
    private boolean enableSoundEffects;
    private boolean recordData;
}

```

```

private boolean recordCarData;
private boolean recordObstacleData;
// private boolean record
private boolean useSimulatedVelocity; //or used dyno velocity
with simulated side slip to determine velocity components

private JTextField filePath;
private JPanel buttonPanel;
private JButton setFileButton;

private float lidarNoise;

// private JFileChooser fileChooser;

public SimulationSettings() {
    limitFPS = false;
    enableSoundEffects = true;

    setBackground(SimData.CalPoly_CREAM);

    GridLayout buttonLayout = new GridLayout(1,0);
    //Set up the horizontal gap value
    buttonLayout.setHgap(25);
    buttonPanel = new JPanel();
    buttonPanel.setBackground(SimData.CalPoly_CREAM);

    buttonPanel.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    // buttonPanel.setSize(new Dimension(300,180));
    buttonPanel.setLayout(buttonLayout);
    createSetFileButton();
    buttonPanel.add(setFileButton);

    add(buttonPanel);
}

@Override
public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();
    System.out.println(e.getSource());
    System.out.println(action);
    if(action.equals("File")) {
        createJFileChooser();
    }
}

private void createJFileChooser() {
    final JFileChooser fileChooser = new JFileChooser();
    ActionListener fileListener = new ActionListener() {

```

```

        @Override
        public void actionPerformed(ActionEvent ae) {
            //Process action performed
            String action = (String)
ae.getActionCommand();
                System.out.println( action );
                if( action == "ApproveSelection")
                    System.out.println( (String)
fileChooser.getSelectedFile().getAbsolutePath() );

                remove( fileChooser );
            }
        };
        fileChooser.addActionListener( fileListener );
        add( fileChooser );
        validate();
    }
    private void createSetFileButton() {
        setFileButton = new JButton("FILE...");
        setFileButton.setVerticalTextPosition(AbstractButton.CENTER);
        setFileButton.setHorizontalTextPosition(AbstractButton.CENTER);
        setFileButton.setMnemonic('s');
        setFileButton.setActionCommand("File");
        setFileButton.addActionListener(this);
    }
}
/**  \class EmbeddedSettings
 *  \brief EmbeddedSettings class is responsible for handling and passing
all Embedded Setting GUI parameters
 *
 *          to the simulation
 */
public class EmbeddedSettings extends JPanel implements ActionListener
{
    private String COMPORT;
    private JComboBox<String> comPortSelect;

    public EmbeddedSettings() {
        setBackground(SimData.CalPoly_CREAM);
        String[] serialPorts =
SerialCommunicationJSSC.getSerialPorts();
        try {
            COMPORT = serialPorts[0];
        } catch(Exception except) {}
    }
}

```

```

        comPortSelect = new JComboBox<String>(serialPorts);
        comPortSelect.setLightWeightPopupEnabled(false);
        comPortSelect.setOpaque(true);
        add( new JLabel("Embedded Device Settings") );
        add( new JLabel("Serial COM Port Selection:") );
        add( comPortSelect );

        comPortSelect.addActionListener( this );

        this.setOpaque(false);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String portName = (String)cb.getSelectedItem();
        System.out.println("Selected port: "+portName);
        COMPORT = portName;
    }
    public String getSerialPort() {
        return COMPORT;
    }
}
private class ColorComboBoxEditor extends JButton implements
ListCellRenderer {
    protected DefaultListCellRenderer defaultRenderer = new
DefaultListCellRenderer();

    public ColorComboBoxEditor() {
        setOpaque(true);
    }

    public Component getListCellRendererComponent(JList list,
Object value, int index, boolean isSelected, boolean cellHasFocus) {
        JLabel renderer = (JLabel)
defaultRenderer.getListCellRendererComponent(list, value, index, isSelected,
cellHasFocus);
        if (value instanceof Color) {
            renderer.setBackground((Color) value);
        }
        return renderer;
    }
}

```

UCS/SimData.java

```

/**
 *      \file SimData.java
 *      \brief Represents the data or data interface of the application to draw and update.
 *
 *      Revisions:
 *
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package UCS;
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
```

```

import java.awt.image.BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, Serializable, Logging, and other
utilities)
import java.util.Random; // Provides pseudo-random generator
import java.util.concurrent.*; // Provides thread concurrency
import java.io.*; // Provides serializable
import java.util.logging.FileHandler;
import java.util.logging.Formatter;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
import java.util.*;
import java.text.*;
// Joystick API Imports
import at.wisch.joystick.*;
import at.wisch.joystick.event.*;
import at.wisch.joystick.exception.*;
import at.wisch.joystick.ffeffect.*;
import at.wisch.joystick.ffeffect.direction.*;

/**
 * \class SimData
 * \brief Represents the data or data interface of the application.
 *
 */
public class SimData implements Serializable {
    /**
     * Singleton instance of SimData class in order to comply with singleton java
     design pattern */
    private static SimData simData = null;
    private final static Dimension screenSize =
    Toolkit.getDefaultToolkit().getScreenSize();
    /**
     * Serializable UID for saving/loading functionality */
    private static final long serialVersionUID = 7989078461104748964L;
    /**
     * Variable to hold the SettingsData of the application and simulation including
     graphics settings,
     * controller settings, embedded hardware settings, etc...
     */
    private SettingsData settingsData = new SettingsData();
    public static final Color CalPoly_GOLD = new Color(122,91,17);
    public static final Color CalPoly_GREEN = new Color(2,73,48);
    public static final Color CalPoly_CREAM = new Color(214,204,175);
    private SerialCommunicationJSSC embeddedHardware = null;
    protected Thread processingThread;
}

```

```

protected boolean processingData = false;
///< Boolean to indicate if the Beaglebone embedded hardware is currently
processing data (a point-cloud)
protected static int RRTindicatorLEDcolor = 1;
/** True if the simulation is currently "running", i.e. the sim loop is looping */
private static boolean running = false;
/** True if the simulation is currently "running", i.e. the sim loop is looping */
private static boolean paused = true;
/** True if the simulation is under joystick control */
private static boolean joystickControl = false;
/** True if joystick has force feedback capabilities */
private static boolean isJoystickFF = false;
/** Milliseconds between updates when not running the sim at max speed */
private static final long slowWaitTimeBetweenUpdates = 100;
/** Milliseconds between updates when running the sim at max speed,
in this case, no waiting at all with a time of 0 */
private static final long fastWaitTimeBetweenUpdates = 20;
/** Represents the known bounds of the sim's area */
/** Milliseconds between updates when not running the sim at max speed */
private static final long slowWaitTimeBetweenRenders = 20;
/** Milliseconds between updates when running the sim at max speed,
in this case, no waiting at all with a time of 0 */
private static final long fastWaitTimeBetweenRenders = 0;
/** Represents the known bounds of the sim's area */
private float STEER_LIMIT = 17.0f;
///< Mechanically imposed steering limit
private float steerGearReduct = 12.0f;
private Rectangle boundedSimArea;
// VDHILS Simulation Object */
private HardwareInLoopSimulation simulation;
/** Holds the simulation run time */
private static double simTime;
/** Delay controlling the update speed */
private long currentUpdateSpeed;
/** Delay controlling the update speed */
private long currentRenderSpeed;
/** Previous Time */
private long oldTimeUpdate,oldTimeRender;
/** Holds the elasped time FPS was calculated */
private long timeSinceLastFPSCalculation = 0;
/** Holds the elasped time UPS was calculated */
private long timeSinceLastUPSCalculation = 0;
private float steerangle = 0;
/** The number of frames */
private int frames = 0;
/** The number of simulation updates */

```

```

private int updates = 0;
/** Holds the latest calculated value of frames per second */
private int fps = 0;
/** Holds the latest calculated value of updates per */
private int ups = 0;
    /** Holds reference to the joystick */
    private transient static FFJoystick joystick;
    /** Joystick effect */
    private static Effect eff;
    /** VDHILS Logger */
    public static final Logger logger = Logger.getLogger("UCS LOG");
    /** Filehandler for the saving/loading of serialized object operations */
    private static FileHandler fh;
    /** Holds reference to input enabled flag */
    protected static boolean inputEnabled = true;

    /**
     * Constructs a new SimData. SimData's constructor creates and
     * schedules an initial updating task
     *
     * \param width The width of the bounded sim area
     * \param height The height of the bounded sim area
     */
    public SimData(int width, int height) {
        simTime = 0.0;
        // Define the know simulation area
        boundedSimArea = new Rectangle(width, height);
        // Initialize random object to obtain psuedo-random numbers
        //Random random = new Random();
        try {
            Date dateNow = new Date();
            SimpleDateFormat dateNowFormater = new
SimpleDateFormat("yyMMddHHmmssZ");
            fh = new FileHandler("./logs/LogFile-
"+dateNowFormater.format(dateNow)+".log");
            logger.addHandler(fh);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);
            logger.setUseParentHandlers(false);
            logger.info("LOGFILE CREATED");
        } catch(SecurityException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }
        // embeddedHardware = new SerialCommunicationJSSC();
    }
}

```

```

        // embeddedHardware.connect();

        // processingThread = (new Thread(embeddedHardware));

        // Initialize the time, fps, and other variables
currentUpdateSpeed = 50;
        // currentUpdateSpeed = fastWaitTimeBetweenUpdates;
        currentRenderSpeed = 10;
        // currentRenderSpeed = fastWaitTimeBetweenRenders;
        // Initialize other parameters explicitly
oldTimeUpdate = System.nanoTime();
        oldTimeRender = System.nanoTime();
    }

    /**
     * Method to allow global access to single instance of SimData class
     * following the java singleton design pattern
     */
    public synchronized static SimData getInstance() {
        if(simData == null) {
            // simData = new SimData();
        }
        return simData;
    }

    public synchronized void writeDataToFile(String filepath) {
        try {
            // Serialize data object to a file
            ObjectOutputStream DataOut = new ObjectOutputStream(new
FileOutputStream(filepath+".ser"));
            DataOut.writeObject(this);
            DataOut.close();

            //Serialize data object to a byte array
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            DataOut = new ObjectOutputStream(bos);
            DataOut.writeObject(this);
            DataOut.close();
            byte[] buf = bos.toByteArray();
            SimData.logger.info("Save data successfully");
        } catch (IOException e) {
            SimData.logger.warning("Failed to write data object to file (" +
e.toString() + ")");
        }
    }

    /**
     * Resets the last known time of the last update,
     * useful for when there has not been an update in a long time,

```

```

* or when the application first starts where there may have been
* some time the application spent doing other things between when
* updating starts and the elaboration of the object.
*/
public void resetTimeOfLastUpdate() {
    oldTimeUpdate = System.nanoTime();
}
    public void resetTimeOfLastRender() {
    oldTimeRender = System.nanoTime();
}
/** Set reference to VDHILS Simulation object
 *
 *      \param sim
 *      \return
 */
public void setSimulation(HardwareInLoopSimulation sim) {
    simulation = sim;
}
public void initializeEmbeddedHardware(String serialPort) {
    embeddedHardware = new SerialCommunicationJSSC(serialPort);
    int attempts = 0;
    boolean connectStatus = false;
    while((attempts<3)&&(connectStatus==false)) {
        connectStatus = embeddedHardware.connect();
        try {
            Thread.sleep(1000);           //1000 milliseconds is one
second.
        } catch(InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
        attempts++;
    }
    if(!connectStatus) {
        System.out.println("Failed to connect...closing application. Please
check serial connections and port #s");
        closeJoystick();
        System.exit(0);
    }
}
public void closeEmbeddedHardware() {
    embeddedHardware.disconnect();
}
public void closeJoystick() {
try{
    if(isJoystickFF) {
        eff.setStrength(0);
}

```

```

                joystick.stopEffect(eff);
                joystick.destroyEffect(eff);
            }
            joystick.stopAll();
            joystick.destroyAll();
            joystick = null;
        } catch(Exception JoystickNull) { logger.warning("No Joystick to
Close"); }
    }

    public synchronized void setJoystick(FFJoystick joystick) {
        this.joystick = joystick;
        eff = joystick.getSimpleEffect();
        eff.setEffectLength(110);
        eff.setStrength(0);
        joystick.newEffect(eff);
        joystick.playEffect(eff,joystick.INFINITE_TIMES);
        settingsData.setJoystick(this.joystick);
    }

    /**
     * Sets the simulation runtime (from class performing numerical integration)
     * \param
     * \return
     */
    public void setSimTime(double simulationTime) {
        simTime = simulationTime;
    }

    /**
     * \return Returns the latest calculated updates per second */
    public static double getSimTime() {
        return simTime;
    }

    /**
     * Retrieves the time between updates to wait
     */
    public long getCurrentWaitTimeBetweenUpdates() {
        return currentUpdateSpeed;
    }

    /**
     * Retrieves the time between updates to wait
     */
    public long getCurrentWaitTimeBetweenRenders() {
        return currentRenderSpeed;
    }

    public void setJoystickControl(boolean joystickControl) {
        this.joystickControl = joystickControl;
    }

    public void setJoystickFF(boolean isJoystickFF) {
        this.isJoystickFF = isJoystickFF;
    }
}

```

```

        }
        public boolean getJoystickControl() {
            return joystickControl;
        }
    /**
     * \return Returns the latest calculated updates per second */
    public int getUPS() {
        return ups;
    }
    /**
     * Set the frames per second */
    public void setFPS(int fps) {
        this.fps = fps;
    }
    /**
     * \return Returns the latest calculated frames per second */
    public int getFPS() {
        return fps;
    }
    /**
     * Start updating the sim data at a slow update speed */
    // public void updateSimDataAtSlowRate() {
    //     currentUpdateSpeed = slowWaitTimeBetweenUpdates;
    // }
    /**
     * Start running the sim data or signal to pause */
    public void setRunning(boolean ToRunOrNotToRun) {
        running = ToRunOrNotToRun;
    }
    /**
     * get paused signal */
    public void setPaused(boolean paused) {
        if(paused)
            inputEnabled = false;
        else
            inputEnabled = true;

        this.paused = paused;
    }
    /**
     */
    public boolean getRunning() {
        return running;
    }
    /**
     * get paused signal */
    public boolean getPaused() {
        return paused;
    }
    /**
     * Start updating the sim data at a fast update speed */
    // public void updateSimDataAtFastRate() {
    //     currentUpdateSpeed = fastWaitTimeBetweenUpdates;
    // }

```

```

/** Updates any objects that need to know how much time has elapsed to update any
needed movements, animations, or events. */
public synchronized void update() {
    // Calculating a new fps/ups value every second
    if (timeSinceLastUPSCalculation >= 1000000000) {
        ups = updates;
        timeSinceLastUPSCalculation = timeSinceLastUPSCalculation -
1000000000;
        updates = 0;
    }
    // Read steering wheel input
    String buttons = "";
    float acceleration = 0;

    float brakes = 0;

    if((running == true)&&(paused==false)&&(embeddedHardware!=null)) {
        if(joystickControl == true) {
            try {
                // Poll joystick for input
                joystick.poll();
                ///\todo Enable user mapping of joystick buttons for
hotkeys (i.e. toggle speedometer, real time plotter, ...)
                // for(int i = 0; i < joystick.getButtonCount(); i++) {
                    // if(joystick.isButtonPressed(i))
                        // System.out.println("Button: "+i+
is pressed");
                // }
                // if (joystick.isButtonPressed(0) &&
joystick.isButtonPressed(1)) {
                    // System.out.println("Buttons 0 and 1 were
pressed at the same time on\njoystick "+joystick.getIndex()+".
Closing program.");
                    //running = false;
                // }
                // Read in joystick poll into appropriate variables
                float f=0;
                for (int a = 0; a < joystick.getAxisCount(); a++) {
                    f = joystick.getAxisValue(a)*100;
                    if(a==0) {
                        acceleration = ((100-f)/200)*100;
                    }
                    else if(a==1) {
                        brakes = ((100-f)/200)*100;
                    }
                    else if(a==3) {

```

```

steerangle =
(float)(f*(Math.PI/180));
}

}

if(joystick.isButtonPressed(14)) {
    // System.out.println("Button Pressed");
    embeddedHardware.setSteerAngle((byte)0);
    embeddedHardware.setThrottle((byte)0);
} else if(joystick.isButtonPressed(19)) {
    embeddedHardware.setSteerAngle((byte)1);
    embeddedHardware.setThrottle((byte)1);
} else {
    //
System.out.println((byte)((steerangle/12.0)*(180/Math.PI)*7.5f+127.67f));
    //
System.out.println((byte)(acceleration*1.27f+128));
    // Convert steer angle in rad to PWM value
for steer servo motor control
    /// \todo allow user-adjustable steer
reduction ratio

    embeddedHardware.setSteerAngle((byte)((steerangle/12.0)*(180/Math.PI)*7.5f+
127.67f));
    // Convert throttle percentage to PWM value
for eletronic speed controller (ESC) for DC brushless motor control

    embeddedHardware.setThrottle((byte)(acceleration*1.27f+128));
}
processingThread = (new
Thread(embeddedHardware));
processingThread.start();

// Set Steering wheel torque
if(eff != null) {
    /// \todo correct steering wheel torque
equation
    //eff.setStrength((int)( -
pow(steerangle,2)*(32767/10000) ));           // Need proper equation -
linear for now
    eff.setStrength((int)( -
steerangle*(32767/100) ));                   // eff.setStrength( 0 );
joystick.updateEffect(eff);
}

```

```

        } catch(Exception JoystickUpdate) {
logger.warning("Joystick Update Failed"); }
}

long elapsedUpdateTime = System.nanoTime() - oldTimeUpdate;
double elapsedTimeSecsUPS =
elapsedTimeUpdate/1000000000.0;
simulation.update( embeddedHardware.getSpeed(), acceleration,
brakes, (float)(embeddedHardware.getSpeed())*elapsedTimeSecsUPS*(1/3600.0));
// Update the simulation if the sim is running. (Perform numerical
integration to get the car states at the next timestep).
simTime = simTime + elapsedTimeSecsUPS;
oldTimeUpdate = oldTimeUpdate + elapsedTimeUpdate;
timeSinceLastUPSCalculation = timeSinceLastUPSCalculation +
elapsedTimeUpdate;
// An update occurred, increment.
updates++;
try{
    processingThread.join();
} catch(Exception e) {}
}

/** This functions sets the steerangle (in rads)
 * \param steerangle Sets the current car steerangle
 */
public void setSteerangle(float steerangle) {
    float reducedSteer = steerangle/steerGearReduct;
    if(Math.abs(reducedSteer) > STEER_LIMIT*(Math.PI/180))
        this.steerangle =
(float)(Math.signum(reducedSteer)*STEER_LIMIT*(Math.PI/180));
    else
        this.steerangle = reducedSteer;
}
/** This function draws all of the renderable simulation data to the graphics context
 * \param drawingBoard 2D graphics context on which to draw
 * \param drawAreaWidth      Width of drawing area
 * \param drawAreaHeight      Height of drawing area
 */
public synchronized void drawSimData(Graphics2D drawingBoard, int
drawAreaWidth, int drawAreaHeight) {
    // Calculating a new fps/ups value every second
    if (timeSinceLastFPSCalculation >= 1000000000) {
        fps = frames;
        timeSinceLastFPSCalculation = timeSinceLastFPSCalculation - 1000000000;
        frames = 0;
    }
}

```

```

}

// This allows our text and graphics to be nice and smooth
drawingBoard.setRenderingHint(
    RenderingHints.KEY_TEXT_ANTIALIASING,
    RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
drawingBoard.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
// Always draw over the image with a blank background, so we
// don't see the last frame's drawings!
drawingBoard.setColor(Color.LIGHT_GRAY);
drawingBoard.fillRect(0, 0, drawAreaWidth, drawAreaHeight);
// Creating a graphics object to not clobber parameter drawingBoard
// where Car's drawing method may change some state of
// the drawingBoard parameter graphics object
Graphics simGraphics = drawingBoard.create();

    if(running == true) {
        simulation.draw(simGraphics);
    }
    // Dispose of the graphics object as were done drawing to it
simGraphics.dispose();

long elapsedTimeRender = System.nanoTime() - oldTimeRender;
double elapsedTimeSecsFPS = elapsedTimeRender/1000000000.0;

oldTimeRender = oldTimeRender + elapsedTimeRender;
timeSinceLastFPSCalculation = timeSinceLastFPSCalculation +
elapsedTimeRender;
        // Increment the number of rendered frames
frames++;
}

public byte getThreat() {
    return embeddedHardware.getThreat();
}
public byte getValidPath() {
    return embeddedHardware.getValidPath();
}
public SettingsData getSettingsData() {
    return settingsData;
}
}

```

UCS/SimUpdater.java

```
/**  
 *      \file SimUpdater.java
```

```

*      \brief SimUpdater handles the high level rendering of the sim at the
BufferStrategy level, as well as handles updating time based events and animations.
*
*      References:
*
*      Revisions:
*          \li 4/6/2013 TFS
*
*      License:
*      This file is copyright 2014 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package UCS;
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
```

```

import java.awt.image.BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, and other utilities)
import java.util.Random;
import java.util.concurrent.*;
import java.util.*;
import java.text.*;
/**  \class SimUpdater
 *   \brief SimUpdater handles the high level rendering of the sim at the
BufferStrategy level, as well as handles updating time based events and animations.
*/
class SimUpdater {
    /** Static reference to instance of simulation data */
    private static SimData simData;

    public SimUpdater(SimData simData) {
        // Set reference
        this.simData = simData;
    }
    /**
     * Simulation update loop run in a separate thread
     */
    public void updateLoop() {
        simData.resetTimeOfLastUpdate();
        // Just loop and loop forever updating the simulation state
        while (true) {
            long nanoTimeAtStartOfUpdate = System.nanoTime();
            // If enough time has passed, update the simulation and all its data
            simData.update();
            // Wait until next update
            waitUntilNextUpdate(nanoTimeAtStartOfUpdate);
        }
    }

    /**
     * Sleeps the current thread if there's still sometime the application
     * can wait for until the time the next update is needed.
     *
     * \param nanoTimeCurrentUpdateStartedOn Time that current update
     *                                         started
     */
    private void waitUntilNextUpdate(long nanoTimeCurrentUpdateStartedOn) {
        // Only sleep to maintain the update speed if speed is higher than

```

```
// zero, because Thread.sleep(0) is not defined on what that
// exactly does
long currentUpdateSpeed = simData.getCurrentWaitTimeBetweenUpdates();
if (currentUpdateSpeed > 0) {
    // This could be more accurate by sleeping what's needed on
    // average for the past few seconds
    long timeToSleep = currentUpdateSpeed - ((System.nanoTime() -
nanoTimeCurrentUpdateStartedOn) / 10000000L);
    // If the speed of updating was so slow that it's time for
    // the next update, then choose 0
    timeToSleep = Math.max(timeToSleep, 0);
    // Again, avoiding Thread.sleep(0)
    if (timeToSleep > 0) {
        try {
            Thread.sleep(timeToSleep);
        } catch (InterruptedException e) {
            // It's okay if we're interrupted, program will just run
            // faster.
            Thread.currentThread().interrupt();
        }
    }
}
}
```

## UCS/Updatable.java

```
package UCS;  
public interface Updatable {  
    public void update(double elaspedTime);  
}
```

## UCS/GUI/MainMenu.java

```
/***
 *      \file MainMenu.java
 *      \brief MainMenu contains GUI components related to manipulating SimData for
 *      to influence SimUpdater object during creation.
 *
 *
 *      \author Thomas Stevens
 *
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
```

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  
 LIMITED TO, THE  
 \* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  
 PARTICULAR PURPOSE  
 \* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  
 CONTRIBUTORS BE  
 \* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
 EXEMPLARY, OR CONSEQUEN-  
 \* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
 SUBSTITUTE GOODS  
 \* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
 INTERRUPTION) HOWEVER  
 \* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,  
 STRICT LIABILITY,  
 \* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY  
 WAY OUT OF THE USE  
 \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
 DAMAGE. \*/

```

//*****
*****  

package UCS.GUI;  

// Import java API's/libraries  

import UCS.SimData;  

// GUI Imports  

import java.awt.*;  

import java.awt.event.*;  

import java.awt.image.*;  

import java.awt.Toolkit;  

import javax.swing.*;  

import javax.swing.border.Border;  

import javax.swing.border.TitledBorder;  

import javax.swing.border.EtchedBorder;  

/**  

 *      \class MainMenu  

 *      \brief MainMenu contains GUI components related to manipulating SimData for  

to influence SimUpdater object during creation.  

 */  

public class MainMenu extends JPanel {  

    private ActionListener MainListener;  

    private int width;  

    private int height;  

    private Color CalPoly_GOLD = new Color(122,91,17);  

    private Color CalPoly_GREEN = new Color(2,73,48);  

    private Color CalPoly_CREAM = new Color(214,204,175);
  
```

```

private JButton startButton,exitButton,SettingsButton;

public MainMenu(ActionListener listener,int width,int height,String versNumber) {
    MainListener = listener;
    this.width = width;
    this.height = height;

    setLayout(new BorderLayout());

    GridLayout buttonLayout = new GridLayout(1,0);
    Border buttonBorder;
    Border blackLine = BorderFactory.createLineBorder(Color.black,3);
    buttonBorder = BorderFactory.createCompoundBorder(blackLine,
    BorderFactory.createEmptyBorder(20,20,20,20));
    //Set up the horizontal gap value
    buttonLayout.setHgap(25);
    JPanel buttonPanel = new JPanel();
    buttonPanel.setBackground(CalPoly_GOLD);
    // buttonPanel.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    buttonPanel.setBorder(buttonBorder);
    // buttonPanel.setSize(new Dimension(300,180));
    buttonPanel.setLayout(buttonLayout);

    createStartButton();
    createSettingsButton();
    createExitButton();

    buttonPanel.add(startButton);
    buttonPanel.add(SettingsButton);
    buttonPanel.add(exitButton);

    add(new SplashScreen("/Resources/car.png",width,height,versNumber));

    add(buttonPanel,BorderLayout.PAGE_END);
    setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

    this.setOpaque(true);
    buttonPanel.setOpaque(true);

    this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
    this.getActionMap().put("Escape pressed", new AbstractAction() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            System.exit(0);
        }
    });
}

```

```

        }
    });

}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
    RenderingHints.VALUE_RENDER_QUALITY);
    int w = getWidth();
    int h = getHeight();
    GradientPaint gradPaint = new GradientPaint(0,0,
    CalPoly_GOLD,width,height, CalPoly_GREEN);
    g2d.setPaint(gradPaint);
    g2d.fillRect(0, 0, w, h);
}

private void createStartButton() {
    startButton = new JButton("START");
    startButton.setVerticalTextPosition(AbstractButton.CENTER);
    startButton.setHorizontalTextPosition(AbstractButton.CENTER);
    startButton.setMnemonic('S');
    startButton.setActionCommand("Start");
    startButton.addActionListener(MainListener);
    startButton.setBackground(CalPoly_CREAM);
    startButton.setToolTipText("Click this button to begin a new simulation");
}

private void createSettingsButton() {
    SettingsButton = new JButton("SETTINGS");
    SettingsButton.setVerticalTextPosition(AbstractButton.CENTER);
    SettingsButton.setHorizontalTextPosition(AbstractButton.CENTER);
    SettingsButton.setMnemonic('T');
    SettingsButton.setActionCommand("Settings");
    SettingsButton.addActionListener(MainListener);
    SettingsButton.setBackground(CalPoly_CREAM);
    SettingsButton.setToolTipText("Click this button to view and/or modify
simulation, graphics, controller, and misc. settings");
}

private void createExitButton() {
    exitButton = new JButton("QUIT");
    exitButton.setVerticalTextPosition(AbstractButton.CENTER);
    exitButton.setHorizontalTextPosition(AbstractButton.CENTER);
    exitButton.setMnemonic('Q');
    exitButton.setActionCommand("Quit");
    exitButton.addActionListener(MainListener);
    exitButton.setBackground(CalPoly_CREAM);
}

```

}

## UCS/GUI/SettingsMenu.java

```

// Modify/View Car button
public class SettingsMenu extends JPanel implements ActionListener {

    private ActionListener MainListener;
    private int width;
    private int height;
    private SimData simData;
    private Color CalPoly_GOLD = new Color(122,91,17);
    private Color CalPoly_GREEN = new Color(2,73,48);
    private Color CalPoly_CREAM = new Color(214,204,175);
    private JButton saveButton,exitButton,backButton,loadButton;
    private final JTabbedPane pane = new JTabbedPane();

    public SettingsMenu(ActionListener listener,int width,int height,SimData
simData) {
        MainListener = listener;
        this.width = width;
        this.height = height;
        this.simData = simData;

        setLayout(new BorderLayout());

        pane.add("Input
Settings",this.simData.getSettingsData().getJoystickSettings());
        pane.add("Embedded Device
Settings",this.simData.getSettingsData().getEmbeddedSettings());

        GridLayout buttonLayout = new GridLayout(1,0);
        //Set up the horizontal gap value
        buttonLayout.setHgap(25);
        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(CalPoly_GOLD);
        Border buttonBorder;
        Border blackLine = BorderFactory.createLineBorder(Color.black,3);
        buttonBorder = BorderFactory.createCompoundBorder(blackLine,
BorderFactory.createEmptyBorder(20,20,20,20));
        buttonPanel.setBorder(buttonBorder);
        buttonPanel.setLayout(buttonLayout);

        createSaveButton();
        createLoadButton();
        createBackButton();
        createExitButton();

        buttonPanel.add(saveButton);

```

```

buttonPanel.add(loadButton);
buttonPanel.add(backButton);
buttonPanel.add(exitButton);

pane.setBorder(blackLine);
pane.setBackground(CalPoly_CREAM);

add(pane,BorderLayout.CENTER);
add(buttonPanel,BorderLayout.PAGE_END);
setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

this.setOpaque(true);
buttonPanel.setOpaque(true);

this.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0, false), "Escape pressed");
this.getActionMap().put("Escape pressed", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        System.exit(0);
    }
});
}

public String getSerialPort() {
    SettingsData.EmbeddedSettings tempSettings =
this.simData.getSettingsData().getEmbeddedSettings();
    return tempSettings.getSerialPort();
}

public void actionPerformed(ActionEvent e) {
    String action = (String) e.getActionCommand();
    System.out.println(e.getSource());
    System.out.println(action);
    if(action.equals("Save")) {
        createJFileChooser();
    }
    else if(action.equals("Load")) {
        createJFileChooser();
    }
}
private void createJFileChooser() {
    final JFileChooser fileChooser = new JFileChooser();
    ActionListener fileListener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
            //Process action performed

```

```

        String action = (String) ae.getActionCommand();
        System.out.println( action );
        if( action == "ApproveSelection")
            System.out.println( (String)
fileChooser.getSelectedFile().getAbsolutePath() );

                remove( fileChooser );
                add(pane);
                validate();
            }
        };
        fileChooser.addActionListener( fileListener );

        remove( pane );
        add( fileChooser );
        validate();
    }
    @Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
    int w = getWidth();
    int h = getHeight();
    GradientPaint gradPaint = new GradientPaint(0,0,
CalPoly_GOLD,width,height, CalPoly_GREEN);
    g2d.setPaint(gradPaint);
    g2d.fillRect(0, 0, w, h);
}
private void createSaveButton() {
    saveButton = new JButton("SAVE");
    saveButton.setVerticalTextPosition(AbstractButton.CENTER);
    saveButton.setHorizontalTextPosition(AbstractButton.CENTER);
    saveButton.setMnemonic('s');
    saveButton.setActionCommand("Save");
    saveButton.addActionListener(this);
    saveButton.setBackground(CalPoly_CREAM);

    saveButton.setEnabled(false);
}
private void createLoadButton() {
    loadButton = new JButton("LOAD");
    loadButton.setVerticalTextPosition(AbstractButton.CENTER);
    loadButton.setHorizontalTextPosition(AbstractButton.CENTER);
    loadButton.setMnemonic('L');
}

```

```

        loadButton.setActionCommand("Load");
        loadButton.addActionListener(this);
        loadButton.setBackground(CalPoly_CREAM);

        loadButton.setEnabled(false);
    }
    private void createBackButton() {
        backButton = new JButton("BACK");
        backButton.setVerticalTextPosition(AbstractButton.CENTER);
        backButton.setHorizontalTextPosition(AbstractButton.CENTER);
        backButton.setMnemonic('B');
        backButton.setActionCommand("BackFromSettings");
        backButton.addActionListener(MainListener);
        backButton.setBackground(CalPoly_CREAM);
    }
    private void createExitButton() {
        exitButton = new JButton("QUIT");
        exitButton.setVerticalTextPosition(AbstractButton.CENTER);
        exitButton.setHorizontalTextPosition(AbstractButton.CENTER);
        exitButton.setMnemonic('Q');
        exitButton.setActionCommand("Quit");
        exitButton.addActionListener(MainListener);
        exitButton.setBackground(CalPoly_CREAM);
    }
}

```

### UCS/GUI/SplashScreen.java

```

/**
 *      \file SplashScreen.java
 *      \brief Splash screen contains GUI components to render splash screen's image
 *
 *      \author Thomas Stevens
 *
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-

```

```

* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  

package UCS.GUI;
import UCS.Graphics.Sprite;
//! Import Libraries
import java.awt.*;
import javax.swing.*;
import java.awt.image.*;

public class SplashScreen extends JPanel {
    private int width;
    private int height;
    private Image image;
    private String versNumber;

    public SplashScreen(String filepath,int width,int height,String versNumber) {
        this.width = width;
        this.height = height;
        this.versNumber = versNumber;

        java.net.URL imgURL = getClass().getResource(filepath);
        BufferedImage mImage;
        try{
            mImage = javax.imageio.ImageIO.read(imgURL);
            int color = mImage.getRGB(0, 0);
            image = Sprite.makeColorTransparent(mImage, new
Color(color));
        } catch(Exception e) {
            System.out.println("Pic error");
            // logText = "Failed to load sprite (" + e.toString() + ")";
            // txtLog.setForeground(Color.red);
            // txtLog.append(logText + "\n");
        }
    }

    this.setBorder(BorderFactory.createEmptyBorder(20,20,20,20));
    this.setOpaque(false);
}

```

```

    }

    public void drawBackground(Graphics2D g) {
        g.drawImage( image,0,0,width,(int)(height*0.85), this);
        g.setFont(new Font("Copperplate Gothic Bold", Font.PLAIN, 40));
        g.setColor(Color.white);
        g.drawString("VDHILS - " + versNumber, 100, 40);
        g.setFont(new Font("Copperplate Gothic Bold", Font.PLAIN, 30));
        g.drawString("Vehicle Dynamics Hardware in the Loop Simulator",
100, 75);
    }
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        drawBackground(g2d);
    }
}

```

UCS/Graphics/SimRenderer.java

```

/**
 *      \file SimRenderer.java
 *      \brief SimRenderer handles the high level rendering of the sim at the
BufferStrategy level, as well as handles updating time based events and animations.
*
*      References:
*
*      Revisions:
*          \li 4/6/2013 TFS
*
*      License:
*      This file is copyright 2014 by T Stevens and released under the Lesser GNU
*      Public License, version 2. It intended for educational use only, but its use
*      is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-

```

\* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  
 OF SUBSTITUTE GOODS  
 \* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
 INTERRUPTION) HOWEVER  
 \* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
 CONTRACT, STRICT LIABILITY,  
 \* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN  
 ANY WAY OUT OF THE USE  
 \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH  
 DAMAGE. \*/  
 //\*\*\*\*\*  
 \*\*\*\*

```

package UCS.Graphics;
// Import java API's/libraries
import UCS.SimData;
import UCS.GUI.*;

// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.awt.Toolkit;
// Error Handling Imports
import java.lang.reflect.InvocationTargetException;
// Utility Imports (Random, Concurrency, Strings/Text, and other utilities)
import java.util.Random;
import java.util.concurrent.*;
import java.util.*;
import java.text.*;
/**  \class SimRenderer
 *   \brief SimRenderer handles the high level rendering of the sim at the
BufferStrategy level
*/
public class SimRenderer implements ResizeListener {
  private SimData simData;
  private BlockingQueue<BufferStrategy> bufferStrategyQueue;
  // Not initialized at creation, but passed in externally when created
  private BufferStrategy bufferStrategy;
  // The component to draw via EDT

```

```

private Component componentToDraw;
private final Rectangle drawAreaBounds;
// We draw almost all graphics to this image, then stretch it over the
// entire frame.
// This allows a resize to make the sim bigger, as opposed to
// just providing a larger area for the sprites to be drawn onto.
// We also are using this image's pixel coordinates as the coordinates
// of our circle sprites.
private BufferedImage drawing;

    public SimRenderer(SimData simData, BlockingQueue<BufferStrategy>
bufferStrategyQueue, int drawingWidth, int drawingHeight) {
        this.simData = simData;
        this.bufferStrategyQueue = bufferStrategyQueue;
        this.drawAreaBounds = new Rectangle(0, 0, drawingWidth, drawingHeight);

        // We draw almost all graphics to this image,
        // then stretch it over the
        // entire frame.
        // This allows a resize to make the sim bigger, as opposed to
        // just providing a larger area for the sprites to be drawn onto.
        drawing = GraphicsEnvironment.getLocalGraphicsEnvironment()
            .getDefauleScreenDevice().getDefauleConfiguration()
            .createCompatibleImage(drawingWidth, drawingHeight);
    }

    public synchronized void setComponentToDraw(Component
componentToDraw) {
        this.componentToDraw = componentToDraw;
    }

    @Override
    public void drawAreaChanged(int x, int y, int width, int height) {
        synchronized (drawAreaBounds) {
            drawAreaBounds.setBounds(x, y, width, height);
        }
    }

    public void renderLoop() {
        // Wait for a buffer strategy from the queue, can't start the sim
        // without being able to draw graphics
        try {
            bufferStrategy = bufferStrategyQueue.poll(1, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            // Thread should not be interrupted, this method is the lowest
            // method the thread should execute from this point on, as the

```

```

        // method has an infinite loop
        e.printStackTrace();
    }
    if (bufferStrategy == null) {
        System.err.println("BufferStrategy could not be made in render loop!");
        System.exit(1);
    }
    // For max accuracy, resetting the time since last update so
    // animations and sprite positions remain in their standard first
    // position
    simData.resetTimeOfLastRender();
    // Just loop and loop forever, update state and then draw.
    while (true) {
        long nanoTimeAtStartOfRender = System.nanoTime();

        try {
            Graphics2D g = (Graphics2D) bufferStrategy.getDrawGraphics();
            drawSim(g);
            g.dispose();
            if (!bufferStrategy.contentsLost()) {
                bufferStrategy.show();
            }
        }
        // This catch is to allow the application to not stop
        // working when the application encounters the possible bug:
        // http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6933331
        // One work around to not encounter this is to Disable d3d
        // using -Dsun.java2d.d3d=false
        // Not sure why the bug is said to "... has no consequences
        // other than a stack trace dump in a console (no hang... ",
        // as people are generally not going to catch an
        // IllegalStateException...
        // You can try to see if you can get the exception to print
        // by resizing the window rapidly on the primary or secondary,
        // or dragging the window off and on the primary monitor.
        // This of course assumes you are using d3d
        catch (IllegalStateException e) {
            e.printStackTrace();
            simData.logger.severe("Illegal state in render loop (" +
e.getStackTrace() + ")");
        }

        waitUntilNextRender(nanoTimeAtStartOfRender);
    }
}

```

```

private synchronized void drawSim(Graphics2D g) {
    // Obtaining the graphics of our drawing image we use,
    // most of the graphics drawn are drawn to this object
    Graphics2D drawingBoard = drawing.createGraphics();
    simData.drawSimData(drawingBoard, drawing.getWidth(),
drawing.getHeight());
    drawingBoard.dispose();
    final Graphics swingAndOtherGuiGraphics = g.create();
    synchronized (drawAreaBounds) {
        // The translate is needed to align our drawing of
        // components to their "clickable" areas (changes where 0, 0
        // actually is, comment it out and see what happens!)
        swingAndOtherGuiGraphics.translate(drawAreaBounds.x,
drawAreaBounds.y);
        Graphics simGraphics = g.create();
        // Image call that scales and stretches the sim's graphics over
        // the entire frame
        // NOTE: This method of stretching graphics is not optimal.
        // This causes a computation of a stretched image each time. A
        // better implementation would be to cache an image of the
        // latest representation of a drawn sim,
        // and re-cache whenever there is a visible change (like color,
        // or its size, which would be due to a window resize), and
        // draw that cached image at the correct calculated location.
        // Additionally, it also causes the rendering to this image to
        // be done on the CPU. See the improvements section in the
        // tutorial.
        simGraphics.drawImage(drawing, drawAreaBounds.x,
            drawAreaBounds.y, drawAreaBounds.width,
            drawAreaBounds.height, null);
        simGraphics.dispose();
    }
    // componentToDraw is lazily set from the EDT during GUI creation or via
    main menu dialog
    if (componentToDraw != null) {
        // Paint our Swing components, to the graphics object of the
        // buffer, not the BufferedImage being used for the
        // application's sprites.
        // We do this, because Swing components don't resize on frame
        // resizes, they just reposition themselves, so we shouldn't
        // stretch their graphics at all.
        try {
            SwingUtilities.invokeAndWait(new Runnable() {
                @Override
                public void run() {
                    if (componentToDraw instanceof JComponent) {

```

```

        ((JComponent) componentToDraw).paintComponents
            (swingAndOtherGuiGraphics);
    } else {
        componentToDraw.paintAll
            (swingAndOtherGuiGraphics);
    }
}
});

} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
} catch (InvocationTargetException e) {
    // should not happen
    e.printStackTrace();
}
}

// In addition, draw the FPS/UPS post stretch, so we always can read
// the info even if you shrink the frame really small.
// Grab the font height to make sure we don't draw the stats outside
// the panel, or over each other.

int fontHeight = g.getFontMetrics(g.getFont()).getHeight();
if(simData.getRunning()) {
    /// \todo draw text within transparent black box like the real
time plotter
    int curFPS = simData.getFPS();
    if(curFPS >= 24)
        swingAndOtherGuiGraphics.setColor(Color.green);
    else if((curFPS>=20)&&(curFPS<24))
        swingAndOtherGuiGraphics.setColor(Color.yellow);
    else
        swingAndOtherGuiGraphics.setColor(Color.red);
    swingAndOtherGuiGraphics.drawString("FPS: " + curFPS, 0,
fontHeight * 4);

    int curUPS = simData.getUPS();
    if(curUPS >= 10)
        swingAndOtherGuiGraphics.setColor(Color.green);
    else if((curUPS>=6)&&(curUPS<10))
        swingAndOtherGuiGraphics.setColor(Color.yellow);
    else
        swingAndOtherGuiGraphics.setColor(Color.red);
    swingAndOtherGuiGraphics.drawString("UPS: " + curUPS, 0,
fontHeight * 5);

    swingAndOtherGuiGraphics.setColor(Color.white);
    DecimalFormat myFormatter = new
DecimalFormat("###,###.##");
}

```

```

        swingAndOtherGuiGraphics.drawString("Time: " +
myFormatter.format(simData.getSimTime()), 0, fontHeight * 6);
    }
    else {
        swingAndOtherGuiGraphics.setColor(Color.white);
        swingAndOtherGuiGraphics.drawString("FPS: " +
simData.getFPS(), 0, fontHeight );
    }
    swingAndOtherGuiGraphics.dispose();
}

/**
 * Sleeps the current thread if there's still sometime the application
 * can wait for until the time the next update is needed.
 *
 * \param nanoTimeCurrentUpdateStartedOn Time that current update
 *                                         started
 */
private void waitUntilNextRender(long nanoTimeCurrentRenderStartedOn) {
    // Only sleep to maintain the update speed if speed is higher than
    // zero, because Thread.sleep(0) is not defined on what that
    // exactly does
    long currentRenderSpeed = simData.getCurrentWaitTimeBetweenRenders();
    if (currentRenderSpeed > 0) {
        // This could be more accurate by sleeping what's needed on
        // average for the past few seconds
        long timeToSleep = currentRenderSpeed - ((System.nanoTime() -
nanoTimeCurrentRenderStartedOn) / 10000000L);
        // If the speed of updating was so slow that it's time for
        // the next update, then choose 0
        timeToSleep = Math.max(timeToSleep, 0);
        // Again, avoiding Thread.sleep(0)
        if (timeToSleep > 0) {
            try {
                Thread.sleep(timeToSleep);
            } catch (InterruptedException e) {
                // It's okay if we're interrupted, program will just run
                // faster.
                Thread.currentThread().interrupt();
            }
        }
        else
            System.out.println("Render too slowly...LAG!");
    }
}
}

```

## UCS/Graphics/Sprite.java

```
/***
 *      \file Sprite.java
 *      \brief Sprite class
 *
 *      Revisions:
 *
 *      License:
 *      This file is copyright 2014 by T Stevens and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto.
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*********************************************************************
*****  
package UCS.Graphics;  
import UCS.SimData;  
// Import Libraries  
import java.awt.*;  
import java.awt.image.*;  
import javax.swing.ImageIcon;  
import java.util.ArrayList;  
import java.io.*;  
import javax.swing.*;  
/*** \class Sprite
```

```

*           \brief Sprite class
*/
public abstract class Sprite implements Serializable, Drawable {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to
the serialized output stream */
    private static final long serialVersionUID = 7989078461104748971L;
    protected Dimension screenSize;    /**< Monitor's screen dimensions in
pixels */
    public float    spriteScale;    /**< Sprite's scale for rendering to screen */
    public int      spriteWidth;   /**< Sprite's image width in pixels */
    public int      spriteHeight;  /**< Sprite's image height in pixels */
    /** List of sprite image filepaths - sprite may consist of multiple sub-sprites
*/
    protected ArrayList<String> imagestr = new ArrayList<String>();
    /** List of sprite image's loaded from image filepaths - sprite may consist of
multiple sub-sprites */
    protected ArrayList<ImageIcon>  image = new ArrayList<ImageIcon>();
    protected boolean   visible;        /**< Boolean indicating if sprite is
visible */
    protected float     depth = 1.0f;   /**< Depth value indicating sprite's alpha
value (transparency) */
    protected String    filePath;       /**< String holding the filepath to
the root sprite image */
    protected boolean   transparent;   /**< Boolean indicating if sprite's
background should be made transparent */
    public abstract void draw(Graphics g);

    public Sprite( Dimension screen, float scale, String filePath, boolean
transparent) {
        spriteScale = scale;

        addImage( filePath, transparent );

        screenSize = screen;
        visible = true;
        setVisible(true);
        this.filePath = filePath;
        this.transparent = transparent;
    }
    // No argument constructor for serializable operations (First non-serializable
class must have a no argument constructor for deserialization)
    public Sprite() {
        //      addImage( filePath, transparent );
    }
}

```

```

protected void addImage(String imagePath,boolean transparent) {
    if( !(imagePath == null || imagePath.trim().equals("")) ) {
        imagestr.add( imagePath );
        java.net.URL imgURL = getClass().getResource(imagePath);
        BufferedImage mImage;
        try{
            mImage = javax.imageio.ImageIO.read(imgURL);
            int color = mImage.getRGB(0, 0);
            spriteWidth = mImage.getWidth();
            spriteHeight = mImage.getHeight();
            if(transparent == true)
                image.add( new
                    ImageIcon(makeColorTransparent(mImage, new Color(color))) );
            else
                image.add( new ImageIcon((Image) mImage));
        } catch(Exception e) {
            System.out.println("Pic error: "+e.toString() + "
"+imagePath);
            SimData.logger.warning("Failed to load sprite
imagePath due to " + e.toString());
        }
    }
}
/***
 *      Function to return value indicating whether sprite is visible
 *      \return visible
 */
public boolean isVisible() {
    return visible;
}
// /**
// * Function to set value indicating whether sprite is visible
// */
protected void setVisible(boolean visibility) {
    visible = visibility;
}
/***
 *      Function to get sprite's scale
 *      \return spriteScale
 */
public float getSpriteScale() {
    return spriteScale;
}
/**
 *
 */

```

```

        public AlphaComposite makeComposite() {
            int type = AlphaComposite.SRC_OVER;
            return(AlphaComposite.getInstance(type,depth));
        }
        /**
         *
         */
        public Image getImage(int index) {
            return image.get(index).getImage();
        }
        /**
         *
         */
        public Image getImage() {
            return image.get(0).getImage();
        }
        /**
         *
         */
        public static Image makeColorTransparent(BufferedImage im,final Color
color) {
            ImageFilter filter = new RGBImageFilter() {
                public int markerRGB = color.getRGB() | 0xFF000000;
                public final int filterRGB(int x,int y,int rgb) {
                    if ((rgb | 0xFF000000) == markerRGB) {
                        // Mark the alpha bits as zero - transparent
                        return 0x00FFFFFF & rgb;
                    } else {
                        // nothing to do
                        return rgb;
                    }
                }
            };
            ImageProducer ip = new FilteredImageSource(im.getSource(),filter);
            return Toolkit.getDefaultToolkit().createImage(ip);
        }
    }
}

```

### UCS/Graphics/GuiCreatorRunnable.java

```

/**
 *      \file GuiCreatorRunnable.java
 *      \brief GuiCreatorRunnable is a runnable, that creates the GUI and a
BufferStrategy, and stores the reference to the strategy to later retrieve.
 *
 *      References:

```

```

*
*   Revisions:
*       \li 4/6/2014 TFS
*
*   License:
*   This file is copyright 2014 by T Stevens and released under the Lesser GNU
*   Public License, version 2. It intended for educational use only, but its use
*   is not limited thereto.
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.*/
//*****
*****
```

```

package UCS.Graphics;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.awt.image.BufferStrategy;
import java.awt.image	BufferedImage;
import java.lang.reflect.InvocationTargetException;
import java.util.Random;
import java.util.concurrent.*;
import java.awt.Toolkit;
```

```

/**  \class GuiCreatorRunnable
 *   \brief GuiCreatorRunnable is a runnable, that creates the GUI and a
BufferStrategy, and stores the reference to the strategy to later retrieve.
 */
public class GuiCreatorRunnable implements Runnable {
    private int width;
    private int height;
    protected SimRenderer simRenderer;
    private JPanel panelContent;
    private BlockingQueue<BufferStrategy> bufferStrategyQueue;
    private JFrame frame;
    private String versNumber;
    /**
     * Constructs a GuiCreatorRunnableFuture, with the requested width
     * and height, and the SimRenderer object to update with a reference
     * to a SwingComponentDrawer, a SimData object to pass to the GUI controls,
     * and the BlockingQueue to store a BufferStrategy
     */
    public GuiCreatorRunnable(JFrame frame, int width, int height, SimRenderer
simRenderer, BlockingQueue<BufferStrategy> bufferStrategyQueue, JPanel
panelContent, String versNumber) {
        this.frame = frame;
        this.width = width;
        this.height = height;
        this.simRenderer = simRenderer;
        this.bufferStrategyQueue = bufferStrategyQueue;
        this.panelContent = panelContent;
        this.versNumber = versNumber;
    }
    @Override
    public void run() {
        frame.setTitle("User Control Station - " + versNumber);
        // Ignore repaints, as we will actively render the frame's graphics
        // ourselves
        frame.setIgnoreRepaint(true);
        // While we have the frame reference available, set it's content
        // pane to not be opaque.
        // The JFrame's content pane's background would otherwise paint over
        // any other graphics we painted ourselves
        if (frame.getContentPane() instanceof JComponent) {
            // JComponent's have a handy setOpaque method
            ((JComponent) frame.getContentPane()).setOpaque(false);
        } else {
            frame.getContentPane().setBackground(new Color(0, 0, 0));
        }
    }
}

```

```

    }

    GraphicsEnvironment env =
GraphicsEnvironment.getLocalGraphicsEnvironment();
    GraphicsDevice device = env.getDefaultScreenDevice();
    boolean isFullScreen = false;//device.isFullScreenSupported();
    // isFullScreen = false;
    //frame.setUndecorated(isFullScreen);
    // frame.setUndecorated(true);
    frame.setResizable(true);
    // if (isFullScreen) {
        // Full-screen mode
        // device.setFullScreenWindow(frame);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // frame.validate();
    // } else {
        // Windowed mode
        frame.pack();
        frame.setVisible(true);
    // }

    // Change width and height of window so that the available
    // screen space actually corresponds to what is passed, another
    // method is the Canvas object + pack()
    frame.setSize(width, height);
    Insets insets = frame.getInsets();
    int insetWide = insets.left + insets.right;
    int insetTall = insets.top + insets.bottom;
    frame.setSize(frame.getWidth() + insetWide, frame.getHeight() + insetTall);

    frame.add(panelContent);
    // Create the BufferStrategy, and store the reference to it
    frame.createBufferStrategy(2);
    try {
        bufferStrategyQueue.put(frame.getBufferStrategy());
    } catch (InterruptedException e) {
        // Should not be interrupted
        e.printStackTrace();
    }
    // SimUpdater will render all of the frame's components
    simRenderer.setComponentToDraw(frame.getContentPane());
}
}

```

UCS/Graphics/JFrameWithResizeListener.java

```

/**
 *      \file JFrameWithResizeListener.java
 *      \brief JFrameWithResizeListener is a JFrame that tell's it's sole listener
 *              that it was created with the drawable bounds of it's frame on
resizes.
*
*      \author
*
*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.*/
//*****
*****  

package UCS.Graphics;
// Import java API's/libraries
// GUI Imports
import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
/**
 *      \class JFrameWithResizeListener
 *      \brief JFrameWithResizeListener is a JFrame that tell's it's sole listener
 *              that it was created with the drawable bounds of it's frame on
resizes.
*/
public class JFrameWithResizeListener extends JFrame implements
ComponentListener {

```

```

// Sole listener to the resizes of the frame
private ResizeListener resizeListener;

public JFrameWithResizeListener(ResizeListener resizeListener) {
    this.addComponentListener(this);
    this.resizeListener = resizeListener;
}

@Override
public void componentResized(ComponentEvent e) {
    Insets insets = this.getInsets();
    resizeListener.drawAreaChanged(
        insets.left,
        insets.top,
        this.getWidth() - (insets.left + insets.right),
        this.getHeight() - (insets.top + insets.bottom));
}

@Override
public void componentMoved(ComponentEvent e) {
    // Do nothing, not needed
}

@Override
public void componentShown(ComponentEvent e) {
    // Do nothing, not needed
}

@Override
public void componentHidden(ComponentEvent e) {
    // Do nothing, not needed
}
}

```

### UCS/Graphics/Speedometer.java

```

/**
 *      \file SplashScreen.java
 *
 *      \author Thomas Stevens
 *
 *      MainMenu contains GUI components to render splash screen's image
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE

```

```

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  

package UCS.Graphics;
import UCS.SimData;
//! Import Libraries
import java.awt.*;
import javax.swing.*;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.io.*;
/** \class Speedometer
 * \brief Speedometer class provides the dashboard which includes a
speedometer and the
 * speedometer needle, a debug mileage 'LCD' readout, throttle and
break box meters
 * as well as all the functionality that intuitively goes with these
dashboard items.
*/
/// \bug Save/Load functionality doesn't load all data to objects (i.e. simData and
obstacleAdmin)
/// \bug Paused state disables simulation toolbar buttons
enum State {
    OFF, ON, BLINK
}
public class Speedometer extends Sprite {
    /** Serializable UID for saving/loading functionality */
    /* NOTE: Can declare 'transient' modifier if field/variable is not to be sent to
the serialized output stream */

```

```

private static final long serialVersionUID = 7989078461104748970L;
private float needleAngle;
private final int offset = 25;
private Needle needle;
private float mileage = 0.0f;
public int x_pos;
public int y_pos;
private BoxMeter brakeBoxMeter;
private BoxMeter throttleBoxMeter;
private MileageMeter mileageMeter;
private IndicatorLED rrtIndicator;
private IndicatorLED collisionIndicator;
public boolean mouseOverSpeedometer = false;

private String[] displayMsgs = { "RRT Disabled", "RRT Enabled",
"...running", "...paused", "...error: ", "No Roadbounds" };

public Speedometer(Dimension screen, String filepath, String needlefilepath)
{
    super( screen, 1.0f, filepath, false ); // 0.5 == Scale TODO: add Scale
param
    needle = new Needle( screen, needlefilepath );
    x_pos = (int)((screenSize.getWidth()/2)-
((spriteWidth*spriteScale)/2));
    y_pos = (int)(screenSize.getHeight()-(spriteHeight-
offset)*spriteScale);

    brakeBoxMeter = new BoxMeter("B",0,0.0f,6,44,Color.red);
    throttleBoxMeter = new BoxMeter("T",1,0.0f,6,44,Color.green);
    mileageMeter = new MileageMeter(
x_pos+(spriteWidth/2)*spriteScale-45f, y_pos+0.80f*spriteHeight*spriteScale,
90.0f, 20.0f, 2,"Courier New");
    rrtIndicator = new IndicatorLED(
x_pos+0.65f*spriteWidth*spriteScale, y_pos+0.85f*spriteHeight*spriteScale, 8f, 0,
State.OFF );
    collisionIndicator = new IndicatorLED(
x_pos+0.325f*spriteWidth*spriteScale, y_pos+0.85f*spriteHeight*spriteScale, 8f, 1,
State.OFF );
}
public synchronized void writeSpeedometerToFile(String filepath) {
    try {
        // Serialize data object to a file
        ObjectOutputStream SpeedoOut = new
ObjectOutputStream(new FileOutputStream(filepath+".ser"));
        SpeedoOut.writeObject(this);
        SpeedoOut.close();
    }
}

```

```

        //Serialize data object to a byte array
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        SpeedoOut = new ObjectOutputStream(bos);
        SpeedoOut.writeObject(this);
        SpeedoOut.close();
        byte[] buf = bos.toByteArray();
        SimData.logger.info("Save speedometer successfully");
    } catch (IOException e) {
        SimData.logger.warning("Failed to write speedometer object to
file (" + e.toString() + ")");
    }
}
protected void toggleRRTIndicator() {
    rrtIndicator.toggleState();
}
protected void setRRTIndicator(int RRTIndicator) {
    if(RRTIndicator==1)
        rrtIndicator.setState(State.ON);
    else
        rrtIndicator.setState(State.OFF);
}
protected void toggleCIndicator() {
    collisionIndicator.toggleState();
}
protected void setCIndicator(int CIndicator) {
    if(CIndicator==1)
        collisionIndicator.setState(State.ON);
    else
        collisionIndicator.setState(State.OFF);
}
public void draw(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    Composite c = g2d.getComposite();
    if(mouseOverSpeedometer)

        g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER
, 0.3f));
        if(image.size() > 0)
            g2d.drawImage(
getImage(),x_pos,y_pos,(int)(spriteWidth*spriteScale),(int)(spriteHeight*spriteScale
), null);

        throttleBoxMeter.draw(g2d);
        brakeBoxMeter.draw(g2d);
        mileageMeter.draw(g2d);
}

```

```

        rrtIndicator.draw(g2d);
        collisionIndicator.draw(g2d);
        needle.draw(g2d);

        g2d.setComposite(c);
    }

    public synchronized void update(float speed, float throttleMeter, float
brakeMeter, double deltaMileage, int RRTIndicator, int CIndicator) {
        needle.update(speed);
        mileage += (float)deltaMileage;
        throttleBoxMeter.update(throttleMeter);
        brakeBoxMeter.update(brakeMeter);
        setRRTIndicator(RRTIndicator);
        setCIndicator(CIndicator);
    }

    private class Needle extends Sprite {
        private int needle_x_pos;
        private int needle_y_pos;
        private float needleAngle;
        private final float radPerMPH = 0.2425f/10.0f;
        private float zeroAngle = -79.25f*radPerMPH;
        public Needle(Dimension screen, String filepath) {
            super( screen, 1.0f, filepath, true); // 0.5 == Scale TODO: add
Scale param
            needle_x_pos = (int)((screenSize.getWidth()/2)-
((spriteWidth*spriteScale)/2));
            needle_y_pos = (int)(screenSize.getHeight()-(spriteHeight-
offset)*spriteScale);
            needleAngle = zeroAngle;
        }
        public void draw(Graphics g) {
            Graphics2D g2d = (Graphics2D) g;
            AffineTransform affineTransform = new AffineTransform();
            affineTransform.translate( needle_x_pos,needle_y_pos );
            affineTransform.scale( spriteScale, spriteScale );
            affineTransform.rotate(needleAngle,
spriteWidth/2,0.66f*spriteHeight);
            if(image.size() > 0)
                g2d.drawImage( getImage(), affineTransform, null );
        }
        public synchronized void update(float speed) {
            needleAngle = zeroAngle + speed*radPerMPH;
        }
    }

    private class BoxMeter {

```

```

private String meterName = "";
private float meterValue;
private float meter_xpos;
private float meter_ypos;
private int meterHeight;
private int meterWidth;
private Color meterColor;
public BoxMeter(String meterName,int position,float initialValue,int width, int height,Color meterColor) {
    this.meterName = meterName;
    if(position==0) {
        meter_xpos = (x_pos+(spriteWidth/2)*spriteScale)-(spriteWidth/2)*spriteScale*0.725f;
        meter_ypos = y_pos+0.9f*spr  
iteHeight*spr  
iteScale;
    } else {
        meter_xpos =
(x_pos+(spriteWidth/2)*spr  
iteScale)+(spriteWidth/2)*spr  
iteScale*0.725f;
        meter_ypos = y_pos+0.9f*spr  
iteHeight*spr  
iteScale;
    }
    meterValue = initialValue;
    meterWidth = width;
    meterHeight = height;
    this.meterColor = meterColor;
}
//public BoxMeter() {}
protected void draw(Graphics g) {
    g.setColor(Color.black);
    g.fillRect((int)meter_xpos,(int)(meter_ypos-(12+meterHeight)),meterWidth,meterHeight);
    g.setColor(meterColor);
    g.fillRect((int)(meter_xpos+2),(int)(meter_ypos-10-meterHeight*(meterValue/100.0f)),meterWidth-4,(int)((meterHeight-4)*(meterValue/100.0f)));
    g.drawString(meterName,(int)meter_xpos,(int)meter_ypos);
}
public synchronized void update(float newMeterValue) {
    meterValue = newMeterValue;
}
private class MileageMeter {
    private String[] lineText;
    private int numLines;
    private Color[] lineColor; // \todo Allow filling behind text of different colors to indicate and highlight important information at different levels
    private RoundRectangle2D.Float mileageRect;
    private String fontName;
}

```

```

        public MileageMeter( float x, float y, float rectwidth, float rectheight,
int numLines, String fontName ) {
            mileageRect = new RoundRectangle2D.Float(x,
y,rectwidth,rectheight, 5, 5);
            this.fontName = fontName;
        }
        public void draw(Graphics g) {
            Graphics2D g2d = (Graphics2D) g;
            g2d.setColor( Color.gray );
            g2d.fill( mileageRect );
            g2d.setColor(Color.cyan);
            g2d.setFont( new Font(fontName, Font.BOLD, 8) );
            g2d.drawString("Mileage: "+String.format("%3f",mileage),
(float)mileageRect.getX()+2, (float)mileageRect.getY()+8);
            g2d.drawString("User info here", (float) mileageRect.getX()+2,
(float)mileageRect.getY()+18);
        }
    }
    private class IndicatorLED {
        private float LED_xpos;
        private float LED_ypos;
        private float LED_size;
        private int LED_color;
        private Color[] LED_colors = new Color[] { Color.green, Color.red,
Color.orange };
        private State state; // { off on blink}

        public IndicatorLED(float x, float y, float size, int ledcolor, State
state) {
            LED_xpos = x;
            LED_ypos = y;
            LED_size = size;
            LED_color = ledcolor;
            this.state = state;
        }
        public void draw(Graphics g) {
            // if RRT enabled draw green indicator LED otherwise draw red
indicator LED
            Graphics2D g2 = (Graphics2D) g;

            g2.setColor(new Color(214,163,46));
            g2.fill(new Ellipse2D.Float(LED_xpos-1.5f, LED_ypos-1.5f,
LED_size+3f, LED_size+3f));
            if(state == State.ON)
                g2.setColor(LED_colors[LED_color]);
        }
    }
}

```

```

        else
            g2.setColor(Color.gray);
        g2.fill(new Ellipse2D.Float(LED_xpos, LED_ypos, LED_size,
LED_size));
    }
    protected void setColor(int newColor) {
        if(newColor < LED_colors.length)
            LED_color = newColor;
    }
    protected void setState(State state) {
        this.state = state;
    }
    protected void toggleState() {
        if(state != State.BLINK) {
            if(state == State.ON)
                state = State.OFF;
            else
                state = State.ON;
        }
    }
}
}

```

### UCS/Graphics/WaitLayerUI.java

```

/*
 * \file WaitLayerUI.java
 *
 *
 *      Original Code by:
 *          \author Oracle
 * Modified and Adapted by:
 *          \author Thomas Stevens
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS

```

```

* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  

package UCS.Graphics;  

// Import Libraries  

import javax.swing.*;  

import javax.swing.plaf.LayerUI;  

import java.awt.*;  

import java.awt.event.ActionEvent;  

import java.awt.event.ActionListener;  

import java.beans.PropertyChangeEvent;  

public class WaitLayerUI extends LayerUI<JPanel> implements ActionListener {  

    private boolean mIsRunning;  

    private boolean mIsFadingOut;  

    private Timer mTimer;  

    private int mAngle;  

    private int mFadeCount;  

    private int mFadeLimit = 15;  

    @Override  

    public void paint (Graphics g, JComponent c) {  

        int w = c.getWidth();  

        int h = c.getHeight();  

        // Paint the view.  

        super.paint (g, c);  

        if (!mIsRunning) {  

            return;  

        }  

        Graphics2D g2 = (Graphics2D)g.create();  

        float fade = (float)mFadeCount / (float)mFadeLimit;  

        // Gray it out.  

        Composite urComposite = g2.getComposite();

```

```

        //
g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f *
fade));
        g2.fillRect(0, 0, w, h);
        g2.setComposite(urComposite);

        // Paint the wait indicator.
int s = Math.min(w, h) / 5;
int cx = w / 2;
int cy = h / 2;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setStroke(new BasicStroke(s / 4,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
        g2.setPaint(Color.white);
        g2.rotate(Math.PI * mAngle / 180, cx, cy);
for (int i = 0; i < 12; i++) {
        float scale = (11.0f - (float)i) / 11.0f;
        g2.drawLine(cx + s, cy, cx + s * 2, cy);
        g2.rotate(-Math.PI / 6, cx, cy);
        g2.setComposite(AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER, scale * fade));
    }

    g2.dispose();
}
public void actionPerformed(ActionEvent e) {
    if (mIsRunning) {
        firePropertyChange("tick", 0, 1);
        mAngle += 3;
        if (mAngle >= 360) {
            mAngle = 0;
        }
        if (mIsFadingOut) {
            if (--mFadeCount == 0) {
                mIsRunning = false;
                mTimer.stop();
            }
        }
        else if (mFadeCount < mFadeLimit) {
            mFadeCount++;
        }
    }
}

public void start() {

```

```

        if (mIsRunning) {
            return;
        }

        // Run a thread for animation.
        mIsRunning = true;
        mIsFadingOut = false;
        mFadeCount = 0;
        int fps = 24;
        int tick = 1000 / fps;
        mTimer = new Timer(tick, this);
        mTimer.start();
    }

    public void stop() {
        mIsFadingOut = true;
    }

    @Override
    public void applyPropertyChange(PropertyChangeEvent pce, JLayer l)
    {
        if ("tick".equals(pce.getPropertyName())) {
            l.repaint();
        }
    }
}

```

### UCS/Graphics/Drawable.java

```

/*
 *      \file Drawable.java
 *      \brief Drawable interface for drawable objects
 *
 *
 *      \author Thomas Stevens
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 *  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 *  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
```

```

* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****
```

```

package UCS.Graphics;
import java.awt.Graphics;
/** This interface is implemented by each object that can be drawn on screen.
 */
public interface Drawable {
    /** The draw function must be implemented by drawable objects. This
function
        * handles the drawing of the object given the graphics context.
        * \param g Graphics context.
    */
    public void draw(Graphics g);
}
```

UCS/Graphics/ResizeListener.java

```

package UCS.Graphics;
interface ResizeListener {
    public void drawAreaChanged(int x, int y, int width, int height);
}
```

UCS/Graphics/WaitLayerUI.java:

```

/**
* \file WaitLayerUI.java
*
*
* Original Code by:
*     \author Oracle
* Modified and Adapted by:
*     \author Thomas Stevens
```

```

*/
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
* TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
* CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  

package vdhils.Graphics;  

// Import Libraries  

import javax.swing.*;  

import javax.swing.plaf.LayerUI;  

import java.awt.*;  

import java.awt.event.ActionEvent;  

import java.awt.event.ActionListener;  

import java.beans.PropertyChangeEvent;  

public class WaitLayerUI extends LayerUI<JPanel> implements ActionListener {
    private boolean mIsRunning;
    private boolean mIsFadingOut;
    private Timer mTimer;  

    private int mAngle;
    private int mFadeCount;
    private int mFadeLimit = 15;  

    @Override
    public void paint (Graphics g, JComponent c) {
        int w = c.getWidth();
        int h = c.getHeight();

```

```

// Paint the view.
super.paint(g, c);

if (!mIsRunning) {
    return;
}

Graphics2D g2 = (Graphics2D)g.create();

float fade = (float)mFadeCount / (float)mFadeLimit;
// Gray it out.
Composite urComposite = g2.getComposite();
//
g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f *
fade));
g2.fillRect(0, 0, w, h);
g2.setComposite(urComposite);

// Paint the wait indicator.
int s = Math.min(w, h) / 5;
int cx = w / 2;
int cy = h / 2;
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
g2.setStroke(new BasicStroke(s / 4,
BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g2.setPaint(Color.white);
g2.rotate(Math.PI * mAngle / 180, cx, cy);
for (int i = 0; i < 12; i++) {
    float scale = (11.0f - (float)i) / 11.0f;
    g2.drawLine(cx + s, cy, cx + s * 2, cy);
    g2.rotate(-Math.PI / 6, cx, cy);
    g2.setComposite(AlphaComposite.getInstance(
        AlphaComposite.SRC_OVER, scale * fade));
}

g2.dispose();
}
public void actionPerformed(ActionEvent e) {
if (mIsRunning) {
    firePropertyChange("tick", 0, 1);
    mAngle += 3;
    if (mAngle >= 360) {
        mAngle = 0;
    }
}

```

```

        if (mIsFadingOut) {
            if (--mFadeCount == 0) {
                mIsRunning = false;
                mTimer.stop();
            }
        }
        else if (mFadeCount < mFadeLimit) {
            mFadeCount++;
        }
    }

    public void start() {
        if (mIsRunning) {
            return;
        }

        // Run a thread for animation.
        mIsRunning = true;
        mIsFadingOut = false;
        mFadeCount = 0;
        int fps = 24;
        int tick = 1000 / fps;
        mTimer = new Timer(tick, this);
        mTimer.start();
    }

    public void stop() {
        mIsFadingOut = true;
    }

    @Override
    public void applyPropertyChange(PropertyChangeEvent pce, JLayer l)
    {
        if ("tick".equals(pce.getPropertyName())) {
            l.repaint();
        }
    }
}

```

vdhils/Graphics/PlottableSignals.java:

```

package vdhils.Graphics;
import java.awt.*;
public interface PlottableSignals {
    void drawRealTimePlotter(Graphics2D g2d, int plotter_x, int plotter_y);
}

```

}

## Appendix E Post Analysis Tool

### **ADAS\_AVT.cpp**

```
/***
 * \file ADAS_AVT.cpp
 * \brief This is a utility to visualize the results of the segmentation
 * and path-planning algorithms and to create a video with the results.
 * The dependencies include: ffmpeg, gnuplot, and opencv
 * MS Windows was the targeted platform for this software.
 *
 * \author Thomas Stevens
 */
/* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUEN-
 * TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. */
//*****
*****  
#pragma comment(lib, "ws2_32.lib")
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "Segmentation/segmentation_class.h"
#include "RRT/RRT_class.h"
#include <string>
#include <stdlib.h>
#include <iostream>
#include <fstream>
```

```

#include <windows.h>
#include <time.h>
#include <algorithm>
#include <ShellAPI.h>
#include <sstream>
#include <math.h>
#include <stdio.h>
#include <winsock2.h>

#ifndef DEBUG

using namespace std;
using namespace cv;

typedef vector< vector<double> > ArrayList;
typedef vector<double> ArrayDbl;
typedef vector<int> ArrayInt;

struct _pt_cloud {
    double timestamp;
    double *x;
    double *y;
};

struct _car_data {
    double timestamp;
    double steerangle;
    double brake;
    double throttle;
    double WC_x;
    double WC_y;
    double sideslip;
    double yawangle;
    double Vx;
    double Vy;
    double yawrate;
    bool threatECU;
};

struct _statistics {
    int threatEvents;
} statistics;

struct _event_statistics {
    double threatDuration;
    double threatDistance;
};

```

```

        double vEgo;
        double driverThrottle;
        double driverSteer;
        double controllerSteer;
        int numObstacles;
        int numThreatObstacles;
        bool RRT_Success;
        int RRT_iterations;
        threat_type_t threatType;
        threat_region_t threatRegion;
        string image_link;
        string video_link;
    };

time_t analysisStart;
time_t analysisEnd;
int BEAMNUMBER = 0;
bool prevThreat = false;
string gnuplot = "gnuplot -e \"";
string plotscript = "\" Gnuplot\\plot.plt";
string plotcarscript = "\" Gnuplot\\carplot.plt";
string plotcarposscript = "\" Gnuplot\\carplotpos.plt";
string plotcarscript1 = "\" Gnuplot\\carplot1.plt";
string plotcarscript2 = "\" Gnuplot\\carplot2.plt";
string plotcarscript3 = "\" Gnuplot\\carplot3.plt";
// Delimiter (simulation)
string delimiter = " ";
// Delimiter (MATLAB copy and paste)
// string delimiter = "\t";
// bool EXIT_STAT = false;

int connectAndUpdateServer();
// Get current date/time, format is YYYY-MM-DD.HH:mm:ss
const std::string currentDate() {
    time_t now = time(0);
    struct tm tstruct;
    char buf[80];
    tstruct = *localtime(&now);
    // Visit http://en.cppreference.com/w/cpp/chrono/c/strftime
    // for more information about date/time format
    strftime(buf, sizeof(buf), "%Y-%m-%d.%H-%M-%S", &tstruct);

    return buf;
}
int system_no_output( std::string command )
{

```

```

command.insert( 0, "/C " );

SHELLEXECUTEINFOA ShExecInfo = {0};
ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
ShExecInfo.fMask = SEE_MASK_NOCLOSEPROCESS;
ShExecInfo.hwnd = NULL;
ShExecInfo.lpVerb = NULL;
ShExecInfo.lpFile = "cmd.exe";
ShExecInfo.lpParameters = command.c_str();
ShExecInfo.lpDirectory = NULL;
ShExecInfo.nShow = SW_HIDE;
ShExecInfo.hInstApp = NULL;

if( ShellExecuteExA( &ShExecInfo ) == FALSE )
    return -1;

WaitForSingleObject( ShExecInfo.hProcess, INFINITE );

DWORD rv;
GetExitCodeProcess( ShExecInfo.hProcess, &rv );
CloseHandle( ShExecInfo.hProcess );

return rv;
}

void printHelp(char* argv[]) {
    cout<<"Usage: "<< argv[0] << " <pt-cloud file> <car data file (optional)>\n";
}

/**
 * Rotate an image
 */
void rotate(cv::Mat& src, double angle, cv::Mat& dst)
{
    int len = std::max(src.cols, src.rows);
    Point2f pt(len/2., len/2.);
    Mat r = cv::getRotationMatrix2D(pt, angle, 1.0);

    warpAffine(src, dst, r, cv::Size(len, len));
}

int updateProgressBar(int current, int total) {
    // ensure that the file to be downloaded is not empty
    // because that would cause a division by zero error later on
    if (total <= 0) {
        return 0;
    }
    // how wide you want the progress meter to be
    int totaldotz=60;
}

```

```

double fractionDone = (double)current / (double)total;
// part of the progressmeter that's already "full"
int dotz = round(fractionDone * totaldotz);
    // create the "meter"
int ii=0;
printf("%3.0f% % [",fractionDone*100.);
// part that's full already
for ( ; ii < dotz;ii++) {
    printf("=");
}
// remaining part (spaces)
for ( ; ii < totaldotz;ii++) {
    printf(" ");
}
// and back to line begin - do not forget the fflush to avoid output buffering
problems!
printf("] %d/%d\n",current, total);
fflush(stdout);
// if you don't return 0, the transfer will be aborted - see the documentation
return 0;
}
// BOOL CtrlHandler( DWORD fdwCtrlType )
//{
// switch( fdwCtrlType )
//{
//     // Handle the CTRL-C signal.
// case CTRL_C_EVENT:
//         // cout<<endl<<"Cleaning up and exiting..."<<endl;
//         // EXIT_STAT = true;
//         // return(FALSE);
//         // break;
// }
//}
int main(int argc, char* argv[]) {

if ( argc < 2 ) {      // argc should be at least 2 for correct execution
    printHelp(argv);
    return 1;
}

time(&analysisStart);
// Cntrl^C Handler Register
// SetConsoleCtrlHandler( (PHANDLER_ROUTINE) CtrlHandler, TRUE );

statistics.threatEvents = 0;

```

```

ifstream input_file;
input_file.open( argv[1] );
ifstream car_data_file;
car_data_file.open( argv[2] );
ofstream carDataPlot;
carDataPlot.open("cardata.dat", ofstream::out | ofstream::trunc);
ofstream simDataPlot;
simDataPlot.open("simdata.dat", ofstream::out | ofstream::trunc);

cout<<"Running analysis..."<<endl;

_car_data *car_data = NULL;
int carDataPts = 0;
if ( car_data_file.is_open() ) {
    cout<<"Reading car data file... ";
    string carLine;
    while(getline(car_data_file, carLine)) {
        if(carLine != "" || carLine != " " || carLine != "\n")
            carDataPts++;
    }
    car_data_file.close();

    carDataPts -= 1;
    if(carDataPts>0)
        car_data = new _car_data[carDataPts];

    car_data_file.open( argv[2] );
    getline(car_data_file, carLine);
    int carDataCount = 0;
    while(getline(car_data_file, carLine)) {
        car_data[carDataCount].timestamp = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].steerangle = -atof(carLine.substr(0,
carLine.find(delimiter)).c_str())*(180/CV_PI);
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].brake = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].throttle = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].WC_x = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
    }
}

```

```

        car_data[carDataCount].WC_y = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].sideslip = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].yawangle = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].Vx = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].Vy = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].yawrate = atof(carLine.substr(0,
carLine.find(delimiter)).c_str());
        carLine.erase(0,carLine.find(delimiter)+1);
        car_data[carDataCount].threatECU =
(bool)atoi(carLine.substr(0, carLine.find(delimiter)).c_str());
        carDataCount++;
    }
    for(int i=0; i < carDataPts; i++) {
        carDataPlot<<car_data[i].timestamp<<" ";
        carDataPlot<<car_data[i].steerangle<<" ";
        carDataPlot<<car_data[i].brake<<" ";
        if(car_data[i].brake>0)
            carDataPlot<<0.0<<" ";
        else
            carDataPlot<<car_data[i].throttle<<" ";
        carDataPlot<<car_data[i].WC_x<<" ";
        carDataPlot<<car_data[i].WC_y<<" ";
        carDataPlot<<car_data[i].sideslip<<" ";
        carDataPlot<<car_data[i].yawangle<<" ";
        carDataPlot<<car_data[i].Vx<<" ";
        carDataPlot<<car_data[i].Vy<<" ";
        carDataPlot<<car_data[i].yawrate<<" ";
        carDataPlot<<car_data[i].threatECU<<endl;
    }
    car_data_file.close();
    cout<<"Done"<<endl;
}

if ( !input_file.is_open() ) {
    cout<<"Could not open file\n";
    return 1;
}

```

```

} else {
    int line_count = 0;
    int pt_count = 0;
    int imageNumber = 0;
    int carIndex = 0;
    double prevTimeStamp = 0;
    double elaspedTimeSums = 0;
    double elaspedTime = 0;

    Mat steerWheelImg = imread("steering_wheel.png");

    // Get number of points in pt-cloud
    string line; // This will
contain the data read from the file
    getline(input_file, line); // Throw away the
first line
    bool readNumBeams = false;
    int numPtClouds = 0;
    while ( getline(input_file, line) ) {
        if(line == "" || line == "\n") {
            if(readNumBeams==false) {
                readNumBeams = true;
            }
            numPtClouds++;
        }
        else if(readNumBeams==false) { BEAMNUMBER++; }
    }
    BEAMNUMBER -= 1;
    input_file.close();
    #ifdef DEBUG
        cout<<"Number of LiDAR beams: "<< BEAMNUMBER <<
endl;
    #endif
    // Create unique folder to hold output images
    string dateTimeNow = currentDateTime();
    DWORD pid = GetCurrentProcessId();
    char uniqueFilePathBuff[255];

    sprintf(uniqueFilePathBuff,"Output/%s_%d",dateTimeNow.c_str(),pid);
    string uniqueFilePath = uniqueFilePathBuff;
    #ifdef DEBUG
        cout<<"Unique file path: "<<uniqueFilePath<<endl;
    #endif
    string mkdirCommand = "mkdir \\""+uniqueFilePath+"\\";
    system(mkdirCommand.c_str());
}

```

```

        string dataCreatedDate = argv[1];
        dataCreatedDate =
    dataCreatedDate.substr(dataCreatedDate.find_last_of('\\')+1).c_str();
        dataCreatedDate =
    dataCreatedDate.substr(dataCreatedDate.find('_')+1).c_str();
        dataCreatedDate =
    dataCreatedDate.substr(dataCreatedDate.find('_')+1).c_str();
        dataCreatedDate =
    dataCreatedDate.substr(0,dataCreatedDate.find('.')).c_str();
        mkdirCommand = "mkdir
\"" + uniqueFilePath + "/" + dataCreatedDate + "\"";
        system(mkdirCommand.c_str());

        _pt_cloud *pt_cloud;
        pt_cloud = new _pt_cloud;
        pt_cloud->x = new double[BEAMNUMBER];
        pt_cloud->y = new double[BEAMNUMBER];

        ofstream plotdatafile;
        plotdatafile.open("plotdata.dat", ofstream::out | ofstream::trunc);
        ofstream obstacledatafile;
        obstacledatafile.open("ObstacleData.txt", ofstream::out |
ofstream::trunc);
        ofstream roadLinesPlot;
        roadLinesPlot.open("roadlinesplot.txt", ofstream::out |
ofstream::trunc);
        ofstream finalPathPlot;
        finalPathPlot.open("finalpathplot.txt", ofstream::out | ofstream::trunc);
        ofstream finalPathPlot_Lines;
        finalPathPlot_Lines.open("finalpathplotlines.dat", ofstream::out |
ofstream::trunc);
        ofstream finalPathPlot_Lines1;
        finalPathPlot_Lines1.open("finalpathplotlines1.dat", ofstream::out |
ofstream::trunc);
        ofstream failedPathsPlot;
        failedPathsPlot.open("failedPathsplot.dat", ofstream::out |
ofstream::trunc);
        ofstream carPosInVehFrame;
        carPosInVehFrame.open("carPosInVehFrame.dat", ofstream::out |
ofstream::trunc);
        ofstream eventsData;
        eventsData.open(uniqueFilePath + "/" + dataCreatedDate + "/Events.csv",
ofstream::out | ofstream::trunc);
        // eventsData<<"Threat Duration(s),Threat Distance(ft),Num
        Obstacles,Num Threat Obstacles,Threat Type,Threat Region,RRT iterations,RRT
        Fail,ABA Event,Image Link"<<endl;

```

```

eventsData<<"Threat Duration(s),Threat Distance(ft),Num
Obstacles,Num Threat Obstacles,Threat Type,Threat Region,Image Link"<<endl;

    input_file.open( argv[1] );
    getline(input_file, line);                                // Throw away the
first line

    obstacledatafile<<"set object rect from -0.5,-1 to 0.5,1 fs
empty"<<endl;

_event_statistics *event_statistics = new _event_statistics;
event_statistics = new _event_statistics;
event_statistics->threatDuration = 0;
event_statistics->threatDistance = 0;

Segmentation *segmentationAlgorithm = new Segmentation;
cout<<"Processing..."<<endl<<endl;
while ( getline(input_file, line) || input_file.eof() ) { //&&
EXIT_STAT!=true) {

    if(line == "" || line == "\n" || line==" " || line=="\n") {
        // system("cls");
        updateProgressBar(imageNumber, numPtClouds);

        // cout<<"Read point-cloud frame
"<<imageNumber<<endl;

#define DEBUG
        cout<<"Blank line"<<endl<<"Running
Segmentation"<<endl;
#endif
        double vehicleState[2];                                // Vx and Vy
        vehicleState[0] = 0;
        vehicleState[1] = 10;                                  // dummy value ->
used if not valid car data file is passed in
        // Get car data index
        if(car_data != NULL) {
            for(int i = carIndex; i < carDataPts; i++) {
                if(pt_cloud->timestamp <=
car_data[i].timestamp) {
                    carIndex = i;
                    for(int j = 0; j < carDataPts; j++) {
                        double tempX =
(car_data[j].WC_x-car_data[carIndex].WC_x)*cos(car_data[carIndex].yawangle)-
(car_data[j].WC_y-car_data[carIndex].WC_y)*sin(car_data[carIndex].yawangle);

```

```

        double tempY =
(car_data[j].WC_x-
car_data[carIndex].WC_x)*sin(car_data[carIndex].yawangle)+(car_data[j].WC_y-
car_data[carIndex].WC_y)*cos(car_data[carIndex].yawangle);
                                // carPosInVehFrame<<-
(car_data[j].WC_y-car_data[carIndex].WC_y)<<" "<<car_data[j].WC_x-
car_data[carIndex].WC_x<<endl;

carPosInVehFrame<<tempX<<" "<<tempY<<endl;

}

break;
}

vehicleState[1] =
sqrt(pow(car_data[carIndex].Vx,2)+pow(car_data[carIndex].Vy,2))*3.28;

}

// Run segmentation
// cout<<"Running segmentation and threat
algorithm..."<<endl;
segmentationAlgorithm->identifyObstacles(pt_cloud-
>x,pt_cloud->y,pt_cloud->x,pt_cloud->y,pt_cloud->timestamp -
prevTimeStamp,vehicleState, BEAMNUMBER);

ArrayList Obstacle_Array = segmentationAlgorithm-
>getObstacleArray();
ArrayList AllObstacle_Array = segmentationAlgorithm-
>getAllObstacleArray();

for(int i=0; i < Obstacle_Array.size(); i++) {
    vector<double> tempObj = Obstacle_Array[i];
    char obstacleDataBuf[255];
    sprintf(obstacleDataBuf,"set object rect from
%f,%f to %f,%f fc \"red\" fs solid",tempObj[0]-tempObj[2]/2,tempObj[1]-
tempObj[3]/2,tempObj[0]+tempObj[2]/2,tempObj[1]+tempObj[3]/2);
    string obstacleDataStr = obstacleDataBuf;
    obstacledatafile<<obstacleDataStr<<endl;
#ifndef DEBUG
    cout<<"All Obstacle data:
"<<obstacleDataStr<<endl;
    cout<<"Obstacle "<<i<<": ";
    for(int j=0; j < tempObj.size(); j++) {
        cout<<tempObj[j]<< " ";
    }
}

```

```

                cout<<endl;
            #endif
        }

        for(int i=0; i < AllObstacle_Array.size(); i++) {
            vector<double> tempObj = AllObstacle_Array[i];
            char obstacleDataBuf[255];
            sprintf(obstacleDataBuf,"set object rect from
%f,%f to %f,%f fs empty",tempObj[0]-tempObj[2]/2,tempObj[1]-
tempObj[3]/2,tempObj[0]+tempObj[2]/2,tempObj[1]+tempObj[3]/2);
            string obstacleDataStr = obstacleDataBuf;
            obstacledatafile<<obstacleDataStr<<endl;
            #ifdef DEBUG
                cout<<"Obstacle data:
"<<obstacleDataStr<<endl;
                cout<<"Obstacle "<<i<<": ";
                for(int j=0; j < tempObj.size(); j++) {
                    cout<<tempObj[j]<< " ";
                }
                cout<<endl;
            #endif
        }
        // cout<<"Number of obstacles:
"<<AllObstacle_Array.size()<<endl;
        // cout<<"Number of threatening obstacles:
"<<Obstacle_Array.size()<<endl;

        int threat = segmentationAlgorithm->returnThreat();
        threat_type_t threatType = segmentationAlgorithm-
>getThreatType();
        threat_region_t threatRegion = segmentationAlgorithm-
>getThreatRegion();
        // Threat Duration(s), Threat Distance(ft), Num
        Obstacles, Num Threat Obstacles, Threat Type, Threat Region, RRT iterations, RRT
        Fail, ABA Event, Image Link
        if((threat>0) && (prevThreat<1)) {
            // Rising edge
            statistics.threatEvents++;
            prevThreat = 1;
            char buff[100];
            sprintf(buff,"ptcloud%d.png",imageNumber);
            string outputPngasStr = buff;
            event_statistics->image_link =
"=HYPERLINK(\""+outputPngasStr+"\")";
            event_statistics->threatDuration += elaspedTime;
            if(car_data != NULL) {

```

```

        event_statistics->threatDistance +=  

elapsedTime*sqrt(pow(car_data[carIndex].Vx,2)+pow(car_data[carIndex].Vy,2))*3.  

28;  

        event_statistics->threatType = threatType;  

        event_statistics->threatRegion =  

threatRegion;  

        event_statistics->numObstacles =  

AllObstacle_Array.size();  

        event_statistics->numThreatObstacles =  

Obstacle_Array.size();  

    }  

} else if((threat>0) && (prevThreat>0)) {  

    event_statistics->threatDuration += elapsedTime;  

if(car_data != NULL) {  

    event_statistics->threatDistance +=  

elapsedTime*sqrt(pow(car_data[carIndex].Vx,2)+pow(car_data[carIndex].Vy,2))*3.  

28;  

    }  

} else {  

    if(prevThreat>0) {  

        // Falling edge -- write event to file  

        string threatTypeStr;  

switch(event_statistics->threatType) {  

        case NONE:  

            threatTypeStr = "NONE";  

        break;  

        case OBSTACLE:  

            threatTypeStr =  

"OBSTACLE";  

        break;  

        case ROADBOUNDARY:  

            threatTypeStr = "ROAD  

        break;  

        case RRT_MANUEVER:  

            threatTypeStr =  

"MANUEVER";  

        break;  

    }  

    string threatRegionStr;  

switch(event_statistics->threatRegion) {  

    case NA:  

        threatRegionStr = "NONE";  

    break;  

    case LOW:  

        threatRegionStr = "LOW";
    }
}

```

```

        break;
    case MEDIUM:
        threatRegionStr =
    "MEDIUM";
        break;
    case HIGH:
        threatRegionStr = "HIGH";
        break;
    }

    eventsData<<event_statistics-
>threatDuration<<,"<<event_statistics->threatDistance<<,"<<event_statistics-
>numObstacles<<,"<<event_statistics-
>numThreatObstacles<<,"<<threatTypeStr<<,"<<threatRegionStr<<,"<<event_st
atistics->image_link<<endl;
        event_statistics->threatDuration = 0;
        event_statistics->threatDistance = 0;
        event_statistics->image_link = "";
    }
    prevThreat = 0;
}

#endif DEBUG
cout<<"Threat: "<<threat<<endl;
#endif
ArrayDbl Road = segmentationAlgorithm-
>returnRoadBounds();
if(!(Road[0]==0&&Road[1]==0&&Road[2]==0)) {
    double br = -Road[1]/tan(Road[2]*(CV_PI/180));
    double m = -br/Road[1];
    double bl = -Road[0]/tan(Road[2]*(CV_PI/180));

#ifndef DEBUG
    cout<<"Writing to roadline plot
file"<<endl;
#endif

    double x1 = -100;
    double y1 = m*-100+br;
    double x2 = 100;
    double y2 = m*100+br;

    roadLinesPlot<<"set arrow from
"<<x1<<,"<<y1<<" to "<<x2<<,"<<y2<<" nohead ls 4 lw 2 lc rgb
\"black\"<<endl;

```

```

        x1 = -100;
        y1 = m*-100+bl;
        x2 = 100;
        y2 = m*100+bl;

        roadLinesPlot<<"set arrow from
"<<x1<<","<<y1<<" to "<<x2<<","<<y2<<" nohead ls 4 lw 2 lc rgb
\"black\""<<endl;
    }
#endif DEBUG
    cout<<"Road: ";
    for(int i=0; i < Road.size(); i++) {
        cout<<Road[i]<< " ";
    }
    cout<<endl<<endl;
    cout<<"Roadslope left-y intercept right-y
intercept"<<endl;
    cout<<m<<" "<<bl<< " "<<br<<endl<<endl;
#endif
    double coneAngle = segmentationAlgorithm-
>getThreatConeAngle()*(CV_PI/180);
    double thresDist = segmentationAlgorithm-
>getThresholdDistance();
    ofstream threatConePlot;
    threatConePlot.open("threatconeplot.txt", ofstream::out |
ofstream::trunc);
    threatConePlot<<"set arrow from 0,0 to
"<<thresDist<<","<<tan(coneAngle)*thresDist<<" nohead lw 1 lc rgb
\"yellow\""<<endl;
    threatConePlot<<"set arrow from 0,0 to "<<-
thresDist<<","<<tan(coneAngle)*thresDist<<" nohead lw 1 lc rgb
\"yellow\""<<endl;
    threatConePlot.close();

    // Run RRT
    // cout<<"Running RRT path-planning
algorithm..."<<endl;
    bool RRT_Success = false;
    float rrtSteer = 0.0f;
    bool ABA_event = false;
    int ABA_brake = 0;
    if(threat>0) {
        ArrayDbl Vehicle;
        Vehicle.push_back( 0 );
        Vehicle.push_back( 0 );

```

```

        Vehicle.push_back( Road[2] );
                                // Measured/Simulated or
Dummy Yaw Angle
        Vehicle.push_back( vehicleState[1] );
                                // Measured/Simulated or Dummy
Velocity (Speed)
        RRT *rrtAlgorithm = new RRT;

        if(isfinite(Road[0])&&isfinite(Road[1])&&isfinite(Road[2])) {
            rrtAlgorithm->Run_RRT(Vehicle,
Obstacle_Array, Road);
                                // Turnout function to put steer angle as a
function of time in terms of PWM values
            if(rrtAlgorithm->getRRTVerify()) {
                ArrayList safeManuever = rrtAlgorithm-
>getFinalPath();
                #ifdef DEBUG
                    cout<<"Number of pts in
FinalPath: "<<safeManuever.size()<<endl;
                #endif
                for(int i=0; i < safeManuever.size(); i++)
{
                    ArrayDbl safeManuever_step =
safeManuever[i];
                    char obstacleDataBuf[255];
                    sprintf(obstacleDataBuf,"set object
rect from %f,%f to %f,%f fc \"green\" fs solid",safeManuever_step[0]-
0.8333/2.0,safeManuever_step[1]-
1.75/2.0,safeManuever_step[0]+0.8333/2.0,safeManuever_step[1]+1.75/2.0);

                    string obstacleDataStr =
obstacleDataBuf;
                    finalPathPlot<<obstacleDataStr<<endl;
                    if(((safeManuever_step[0]!=0)&&(safeManuever_step[1]>=1.75/2.0))) {
                        sprintf(obstacleDataBuf,"%f
%f",safeManuever_step[0],safeManuever_step[1]);
                        obstacleDataStr =
obstacleDataBuf;
                        finalPathPlot_Lines<<obstacleDataStr<<endl;
                    }
                }
            for(int i=0; i < safeManuever.size(); i++)
{
}

```

```

        ArrayDbI safeManuever_step =
safeManuever[i];
        char obstacleDataBuf[255];

        if(((safeManuever_step[0]!=0)&&(safeManuever_step[1]>=1.75/2.0))) {
            double angle =
safeManuever_step[2]*(CV_PI/180);
            double wb = 0.8333/2.0;
            double x_f =
cos(angle+CV_PI)*wb;
            double y_f =
sin(angle+CV_PI)*wb;

            sprintf(obstacleDataBuf,"%f
%f",safeManuever_step[0]-x_f,safeManuever_step[1]-y_f);
            string obstacleDataStr =
obstacleDataBuf;

            finalPathPlot_Lines1<<obstacleDataStr<<endl;
        }
    }
    for(int i=safeManuever.size()-1; i>0; i--)
{
    ArrayDbI safeManuever_step =
safeManuever[i];
    char obstacleDataBuf[255];

    if(safeManuever_step[1]>=1.75/2.0) {
        double angle =
safeManuever_step[2]*(CV_PI/180);
        double wb = 0.8333/2.0;
        double x_f =
cos(angle+CV_PI)*wb;
        double y_f =
sin(angle+CV_PI)*wb;

        sprintf(obstacleDataBuf,"%f
%f",safeManuever_step[0]+x_f,safeManuever_step[1]+y_f);
        string obstacleDataStr =
obstacleDataBuf;

        finalPathPlot_Lines1<<obstacleDataStr<<endl;
    }
}

```

```

        } else {
            // cout<<"RRT FAILED TO PLAN A
SAFE PATH! -> Mitigate Damage ABA"<<endl;
ABA_event = rrtAlgorithm-
>getABAEvent();
ABA_brake = rrtAlgorithm-
>getABABrake();
}
vector< vector< vector<double> > > failedPaths =
rrtAlgorithm->getFailedPaths();
rrtSteer = rrtAlgorithm->getRRTSteer();
// cout<<"Number of failed paths:
"<<failedPaths.size()<<endl;
RRT_Success = rrtAlgorithm->getRRTVerify();
delete rrtAlgorithm;

if(failedPaths.size()>0) {
    for(int i=failedPaths.size()-1; i > 0; i--) {
        vector< vector<double> > currFailedPath
= failedPaths[i];
        for(int j=0; j<currFailedPath.size(); j++) {
            vector<double> failedpathpoints =
currFailedPath[j];
            if(failedpathpoints[1]>=1.75/2.0) {

failedPathsPlot<<failedpathpoints[0]<<" "<<failedpathpoints[1]<<endl;
}
failedPathsPlot<<endl;
}
}
}
} else {
ABA_event = false;
ABA_brake = 0;
}

// Run plot script to create png
char buff[100];
sprintf(buff,"ptcloud%d.png",imageNumber);
string outputPngasStr = buff;
string outputplotfilename =
"filename='"+uniqueFilePath+"/"+dataCreatedDate+"/"+outputPngasStr+"'";
string gnuplotsyscommand =
gnuplot+outputplotfilename+plotscript;
#endif DEBUG

```

```

cout<<endl<<gnuplotsyscommand<<endl;
#endif
plotdatafile.close();
obstacledatafile.close();
roadLinesPlot.close();
finalPathPlot.close();
finalPathPlot_Lines.close();
finalPathPlot_Lines1.close();
failedPathsPlot.close();
carPosInVehFrame.close();

system_no_output( gnuplotsyscommand.c_str() );

// OpenCV overlay
Mat tempImage =
imread(uniqueFilePath+"/"+dataCreatedDate+"/"+outputPngasStr,
CV_LOAD_IMAGE_COLOR);
// cout<<"Temp img channels =
"<<tempImage.channels()<<endl;
if(tempImage.channels() < 3) {
    cvtColor(tempImage, tempImage,
CV_GRAY2RGB);
}
Mat bigImage = imread("blank.png");
// cout<<"Blank img channels =
"<<bigImage.channels()<<endl;
Rect roi2( Point(100,0), tempImage.size() );
// try {
    // cout<<"First copyTo"<<endl;
    tempImage.copyTo( bigImage(roi2) );
// } catch(...) { cout<<"Failed"<<endl; }
if(car_data != NULL) {
    Mat rotatedSteeringWheelImg;
    rotate(steerWheelImg,
car_data[carIndex].steerangle*10, rotatedSteeringWheelImg);
    Rect roi( Point(0,20),
rotatedSteeringWheelImg.size() );
    // cout<<"Rotated steering wheel 1 channels =
"<<rotatedSteeringWheelImg.channels()<<endl;
    // try{
        // cout<<"Second copyTo"<<endl;
        rotatedSteeringWheelImg.copyTo(
bigImage(roi) );
    // } catch(...) { cout<<"Failed"<<endl; }
}

```

```

        // Draw frame number string
        std::stringstream s_frame;
        s_frame << imageNumber;
        string frameNum = "Frame: "+s_frame.str();
        putText(bigImage,
frameNum,Point(bigImage.size().width*0.475,15),FONT_HERSHEY_SIMPLEX,0.5
,Scalar(0,0,0));

        //Draw Actual data (ECU/measured)
        putText(bigImage, "Sim
Data",Point(bigImage.size().width*0.875,15),FONT_HERSHEY_SIMPLEX,0.5,Scal
ar(0,0,0));
        //Draw Simulated data
        putText(bigImage, "Real
Data",Point(25,15),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,0));

        // Draw red circle is threat is active -- simulation
        if(threat>0) {
            circle(bigImage, Point(bigImage.size().width-
109,157), 10, Scalar(0, 0, 255), -1);
            circle(bigImage, Point(bigImage.size().width-
19,157), 10, Scalar(0, 0, 255), -1);
        }
        else {
            circle(bigImage, Point(bigImage.size().width-
109,157), 10, Scalar(0, 255, 0), -1);
            circle(bigImage, Point(bigImage.size().width-
19,157), 10, Scalar(0, 255, 0), -1);
        }
        // Draw red circle is threat is active -- ECU
        if(car_data[carIndex].threatECU > 0) {
            circle(bigImage, Point(19,157), 10, Scalar(0, 0,
255), -1);
            circle(bigImage, Point(109,157), 10, Scalar(0, 0,
255), -1);
        }
        else {
            circle(bigImage, Point(19,157), 10, Scalar(0, 255,
0), -1);
            circle(bigImage, Point(109,157), 10, Scalar(0,
255, 0), -1);
        }

        putText(bigImage,
"Threat",Point(44,162),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

```

```

        putText(bigImage,
        "Threat",Point(bigImage.size().width-
88,162),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

        std::stringstream s_steerangle_actual;
        std::stringstream s_steerangle_sim;

        float steerangle_actual_f =
car_data[carIndex].steerangle;
        float steerangle_sim_f = rrtSteer;

        // if not equal when threat is high -> highlight
        s_steerangle_actual << car_data[carIndex].steerangle;
        s_steerangle_sim << rrtSteer;
        string steerangle_actual = s_steerangle_actual.str();
        string steerangle_sim;
        Mat rotatedSteeringWheelImg;
        if(threat>0) {
            steerangle_sim = s_steerangle_sim.str();
            rotate(steerWheelImg, rrtSteer*10,
rotatedSteeringWheelImg);
        }
        else {
            steerangle_sim = s_steerangle_actual.str();
            steerangle_sim_f = steerangle_actual_f;
            rotate(steerWheelImg,
car_data[carIndex].steerangle*10, rotatedSteeringWheelImg);
        }
        Rect roi( Point(bigImage.size().width-
rotatedSteeringWheelImg.size().width,20), rotatedSteeringWheelImg.size() );
        // cout<<"Rotated steering wheel 2 channels =
"<<rotatedSteeringWheelImg.channels()<<endl;
        // try{
            // cout<<"Third copyTo"<<endl;
            rotatedSteeringWheelImg.copyTo( bigImage(roi)
);

        // } catch(...) { cout<<"Failed"<<endl; }
        string steerangle_static = "Steer angle (deg):";
        putText(bigImage,
steerangle_static,Point(10,200),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        putText(bigImage,
steerangle_static,Point(bigImage.size().width-
119,200),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

        if(threat>0 && car_data[carIndex].steerangle!=rrtSteer)
{

```

```

                putText(bigImage,
steerangle_actual,Point(10,220),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,153,255)
);
                putText(bigImage,
steerangle_sim,Point(bigImage.size().width-
119,220),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,153,255));
} else {
                putText(bigImage,
steerangle_actual,Point(10,220),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
                putText(bigImage,
steerangle_sim,Point(bigImage.size().width-
119,220),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
}

std::stringstream s_throttle;
std::stringstream s_brake;
std::stringstream s_brake_ABA;
std::stringstream s_speed;

s_throttle << car_data[carIndex].throttle;
s_brake << car_data[carIndex].brake;
s_brake_ABA << ABA_brake;
s_speed << sqrt(pow(car_data[carIndex].Vx,2) +
pow(car_data[carIndex].Vy,2))*2.23693629;

rectangle(bigImage,Point(10,245),Point(112,265),Scalar(0,0,0));
putText(bigImage, "Throttle:
"+s_throttle.str()+"%",Point(10,240),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0)
);

rectangle(bigImage,Point(11,246),Point(11+car_data[carIndex].throttle,264),S
calar(0,255,0, -1);

rectangle(bigImage,Point(bigImage.size().width-
119,245),Point(bigImage.size().width-119+102,265),Scalar(0,0,0));
if(ABA_event) {
putText(bigImage, "Throttle:
0%",Point(bigImage.size().width-
118,240),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
} else {
putText(bigImage, "Throttle:
"+s_throttle.str()+"%",Point(bigImage.size().width-
119,240),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
}

```

```

        rectangle(bigImage,Point(bigImage.size().width-
118,246),Point(bigImage.size().width-
118+car_data[carIndex].throttle,264),Scalar(0,255,0), -1);
    }

rectangle(bigImage,Point(10,285),Point(112,305),Scalar(0,0,0));
putText(bigImage, "Brake:
"+s_brake.str()+"%",Point(10,280),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

rectangle(bigImage,Point(11,286),Point(11+car_data[carIndex].brake,304),Sc
alar(0,0,255), -1);

rectangle(bigImage,Point(bigImage.size().width-
119,285),Point(bigImage.size().width-119+102,305),Scalar(0,0,0));
if(ABA_event) {
    putText(bigImage, "Brake:
"+s_brake_ABA.str()+"%",Point(bigImage.size().width-
119,280),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
    rectangle(bigImage,Point(bigImage.size().width-
118,286),Point(bigImage.size().width-118+ABA_brake,304),Scalar(0,0,255), -1);
} else {
    putText(bigImage, "Brake:
0%",Point(bigImage.size().width-
119,280),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
}

putText(bigImage,
"Speed:",Point(10,320),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
putText(bigImage, s_speed.str()+""
(mph),Point(10,335),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

// Reason for threat
string s_threat_type = "Threat Type: ";
putText(bigImage,
s_threat_type,Point(bigImage.size().width-
119,320),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

if(threatType == NONE) {
    s_threat_type = "None";
} else if(threatType == OBSTACLE) {
    s_threat_type = "Obstacle";
} else if(threatType == ROADBOUNDARY) {
    s_threat_type = "Road Boundary";
} else if(threatType == RRT_MANUEVER) {
    s_threat_type = "Manuever";
}

```

```

        }
        // Threat region -> changes color of string
        if(threatRegion == NA) {
            putText(bigImage,
s_threat_type,Point(bigImage.size().width-
119,335),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        } else if(threatRegion == LOW) {
            putText(bigImage,
s_threat_type,Point(bigImage.size().width-
119,335),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,255,0));
        } else if(threatRegion == MEDIUM) {
            putText(bigImage,
s_threat_type,Point(bigImage.size().width-
119,335),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,153,255));
        } else if(threatRegion == HIGH) {
            putText(bigImage,
s_threat_type,Point(bigImage.size().width-
119,335),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,255));
        }
        // ECU yaw angle
        putText(bigImage, "Yaw Angle
(deg):",Point(10,350),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        std::stringstream s_yaw_ECU;
        double angleDetected = Road[2];
        s_yaw_ECU << angleDetected;
        putText(bigImage,
s_yaw_ECU.str(),Point(10,365),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

        // Sim yaw angle
        std::stringstream s_yaw;
        s_yaw << ((-
car_data[carIndex].yawangle*(180/CV_PI))-90.0)+180.0;
        putText(bigImage, "Yaw Angle
(deg):",Point(bigImage.size().width-
119,350),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        putText(bigImage,
s_yaw.str(),Point(bigImage.size().width-
119,365),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));

        // Number of obstacles
        std::stringstream s_numObs;
        s_numObs << AllObstacle_Array.size();
        std::stringstream s_numThreatObs;
        s_numThreatObs << Obstacle_Array.size();

```

```

        putText(bigImage, "Num
Obs.",Point(bigImage.size().width-
119,380),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        putText(bigImage,
s_numObs.str(),Point(bigImage.size().width-
119,395),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        // Number of threatening obstacles
        putText(bigImage, "Num Threat
Obs.",Point(bigImage.size().width-
119,410),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        putText(bigImage,
s_numThreatObs.str(),Point(bigImage.size().width-
119,425),FONT_HERSHEY_SIMPLEX,0.4,Scalar(0,0,0));
        // Path-planning failure

        // Mileage

imwrite(uniqueFilePath+"/"+dataCreatedDate+"/"+outputPngasStr,
bigImage);

simDataPlot<<car_data[carIndex].timestamp<<
"<<threat<<" "<<steerangle_sim_f<<endl;

imageNumber++;
plotdatafile.open("plotdata.dat", ofstream::out |
ofstream::trunc);
obstacledatafile.open("ObstacleData.txt", ofstream::out |
ofstream::trunc);
obstacledatafile<<"set object rect from -0.5,-1 to 0.5,1 fs
empty"<<endl;
roadLinesPlot.open("roadlinesplot.txt", ofstream::out |
ofstream::trunc);
finalPathPlot.open("finalpathplot.txt", ofstream::out |
ofstream::trunc);
finalPathPlot_Lines.open("finalpathplotlines.dat",
ofstream::out | ofstream::trunc);
finalPathPlot_Lines1.open("finalpathplotlines1.dat",
ofstream::out | ofstream::trunc);
failedPathsPlot.open("failedPathsplot.dat", ofstream::out |
ofstream::trunc);
carPosInVehFrame.open("carPosInVehFrame.dat",
ofstream::out | ofstream::trunc);

line_count = 0;

```

```

        pt_count = 0;
    } else {
        if(line_count == 0) {
            // Parse timestamp
            pt_cloud->timestamp = atof(line.c_str());
            #ifdef DEBUG
                cout<<"timestamp: "<<pt_cloud-
>timestamp<<endl;
            #endif
            elaspedTime = pt_cloud->timestamp -
prevTimeStamp;
            elaspedTimeSums += elaspedTime;
            prevTimeStamp = pt_cloud->timestamp;
        } else {
            // Parse line for x and y pts
            // Rotate (from simulation)
            pt_cloud->y[pt_count] = atof(line.substr(0,
line.find(delimiter)).c_str())*3.28;
            pt_cloud->x[pt_count] =
atof(line.substr(line.find(delimiter)+delimiter.length(), line.length()).c_str())*3.28;
            // Don't rotate (from MATLAB copy and paste)
            // pt_cloud->x[pt_count] = atof(line.substr(0,
line.find(delimiter)).c_str())*3.28;
            // pt_cloud->y[pt_count] =
atof(line.substr(line.find(delimiter)+delimiter.length(), line.length()).c_str())*3.28;
            #ifdef DEBUG
                cout<<"X-pt: "<<pt_cloud-
>x[pt_count]<<" Y-pt: "<<pt_cloud->y[pt_count]<<endl;
            #endif
            plotdatafile<<pt_cloud->x[pt_count]<<"
"<<pt_cloud->y[pt_count]<<endl;
            pt_count++;
        }
        line_count++;
    }

    if(input_file.eof())
        break;
}

delete segmentationAlgorithm;

// Car data plots
if(car_data != NULL) {
    string outputplotfilename =
"filename='"+uniqueFilePath+"/"+dataCreatedDate+"/"+vehicledata.png"+"';
```

```

        string gnuplotsyscommand =
gnuplot+outputplotfilename+plotcarscript;
        system(gnuplotsyscommand.c_str());
        outputplotfilename =
"filename="" +uniqueFilePath+ "/" +dataCreatedDate+ "/" +"vehicledata1.png"";;
        gnuplotsyscommand =
gnuplot+outputplotfilename+plotcarscript1;
        system(gnuplotsyscommand.c_str());
        outputplotfilename =
"filename="" +uniqueFilePath+ "/" +dataCreatedDate+ "/" +"vehiclepos.png"";;
        gnuplotsyscommand =
gnuplot+outputplotfilename+plotcarposscript;
        system(gnuplotsyscommand.c_str());
        outputplotfilename =
"filename="" +uniqueFilePath+ "/" +dataCreatedDate+ "/" +"vehicledata3.png"";;
        gnuplotsyscommand =
gnuplot+outputplotfilename+plotcarscript3;
        system(gnuplotsyscommand.c_str());
    }
    string outputplotfilename =
"filename="" +uniqueFilePath+ "/" +dataCreatedDate+ "/" +"vehicledata2.png"";;
    string gnuplotsyscommand =
gnuplot+outputplotfilename+plotcarscript2;

    system(gnuplotsyscommand.c_str());

// Delete pt_cloud, temp .dat files, and plot files. Close file streams
delete pt_cloud->x;
delete pt_cloud->y;
delete pt_cloud;

delete event_statistics;

plotdatafile.close();
obstaclefile.close();
roadLinesPlot.close();
finalPathPlot.close();
finalPathPlot_Lines.close();
finalPathPlot_Lines1.close();
carDataPlot.close();
simDataPlot.close();
eventsData.close();
failedPathsPlot.close();
carPosInVehFrame.close();
system("rm plotdata.dat");
system("rm ObstacleData.txt");

```

```

        system("rm roadlinesplot.txt");
        system("rm finalpathplot.txt");
        system("rm threatconeplot.txt");
        system("rm finalpathplotlines.dat");
        system("rm finalpathplotlines1.dat");
        system("rm failedPathsplot.dat");
        system("rm cardata.dat");
        system("rm simdata.dat");
        system("rm carPosInVehFrame.dat");

        // Run FFmpeg on output images to create video or use openCV to play
        // through images
        cout<<"Creating video...";
        double avgFrameRate = imageNumber/elapsedTimeSums;
        #ifdef DEBUG
            cout << "Number of frames: "<<imageNumber<<endl;
            cout << "Average Frame Rate:
" <<avgFrameRate<<endl<<endl;
        #endif

        std::stringstream s;
        s << avgFrameRate;
        string makeVideoCommand = "ffmpeg -framerate "+ s.str() +" -i "+
uniqueFilePath+"/"+dataCreatedDate +"/ptcloud%d.png -c:v libx264 -r 30 -pix_fmt
yuv420p "+ uniqueFilePath+"/"+dataCreatedDate +"/out.mp4";
        #ifdef DEBUG
            system(makeVideoCommand.c_str());
        #else
            system_no_output(makeVideoCommand.c_str());
        #endif
        cout<<"Done"<<endl;
        cout<<"Connecting to server... ";
        int serverStat = connectAndUpdateServer();
        if(serverStat >= 0) {
            cout<<"Successfully connected to server"<<endl;
        } else {
            cout<<"Failed to connect to server"<<endl;
        }
        cout<<"Done"<<endl;
        time(&analysisEnd);
        double analysisTime = difftime(analysisEnd,analysisStart);
        cout<<endl<<"Elapsed analysis time: "<<analysisTime<<" s"<<endl;
        cout<<endl<<"Elapsed system time: "<<elapsedTimeSums<<
" s"<<endl;
        cout<<"Number of events: "<<statistics.threatEvents<<endl;
    }
}

```

```

        cout<<endl<<"Analysis complete -- see output files"<<endl;
    }

    delete car_data;

    return 0;
}

int connectAndUpdateServer() {
    const int port = 8080;
    // winsock2 library init
    WSADATA wsa;
    WSAStartup(MAKEWORD(1,1),&wsa);

    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent* server;

    // create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == INVALID_SOCKET)
        return -1;

    server = gethostbyname("localhost");
    serv_addr.sin_family = AF_INET;
    memcpy(&serv_addr.sin_addr.S_un, server->h_addr_list[0], server-
>h_length);
    serv_addr.sin_port = htons(port);

    // try to connect
    if (connect(sockfd, (sockaddr*)&serv_addr, sizeof(serv_addr)) < 0)
        return -1;

    char buf[8192];

    // send request for HTTP server

    strcpy(buf, "123456789");
    send(sockfd, buf, strlen(buf), 0);

    // receive data
    // c - number of received bytes

    // int c = recv(sockfd, buf, strlen(buf), 0);
    // close socket

    closesocket(sockfd);
}

```

```
}
```

```
Gnuplot/carplot.plt
```

```
set terminal png font arial 14 size 800,600
set output filename
set multiplot layout 2, 1
set xlabel "Time(s)"
set grid
unset key
set ylabel "Throttle(%) and Brake(%)"
plot "./cardata.dat" using 1:4 lt rgb "green" with lines, " using 1:3 lt rgb "red" with
lines
set ylabel "Velocity(m/s)"
plot "./cardata.dat" using 1:9 lt rgb "blue" with lines, " using 1:10 lt rgb "green" with
lines
unset multiplot
```

```
Gnuplot/carplot1.plt
```

```
set terminal png font arial 14 size 800,600
set output filename
set multiplot layout 2, 1
set xlabel "Time (s)"
set grid
unset key
set yrangle[-1.5:1.5]
set ylabel "Controller Override (ECU)"
plot "./cardata.dat" using 1:12 lt rgb "blue" with lines
set ylabel "Controller Override (Sim)"
plot "./simdata.dat" using 1:2 lt rgb "blue" with lines
unset multiplot
```

```
Gnuplot/carplot2.plt
```

```
set terminal png font arial 14 size 800,600
set output filename
set multiplot layout 2, 1
set xlabel "Time(s)"
set grid
unset key
set ylabel "Steerangle ECU(deg)"
plot "./cardata.dat" using 1:2 lt rgb "blue" with lines
set ylabel "Steerangle Sim(deg)"
plot "./simdata.dat" using 1:3 lt rgb "blue" with lines
unset multiplot
```

Gnuplot/carplot3.plt

```
set terminal png font arial 14 size 800,600
set output filename
set multiplot layout 2, 1
set xlabel "Time(s)"
set grid
unset key
set ylabel "Yaw Angle(rad)"
plot "./cardata.dat" using 1:8 lt rgb "blue" with lines
set ylabel "Yaw Rate (rad/s)"
plot "./cardata.dat" using 1:11 lt rgb "blue" with lines
unset multiplot
```

Gnuplot/carplotpos.plt

```
set terminal png font arial 14 size 800,600
set output filename
set xlabel "X Position(ft)"
set ylabel "Y Position(ft)"
set grid
unset key
set size ratio -1
plot "./cardata.dat" using 5:6 lt rgb "blue" with lines
```

Gnuplot/plot.plt

```
set terminal png font arial 14 size 800,600
set output filename
#set xlabel "X Position(ft)"
#set ylabel "Y Position (ft)"
set grid
unset key
set xrange[-20:20]
set yrange[-15:20]
set size ratio -1
load "ObstacleData.txt"
load "roadlinesplot.txt"
# Road lines plot -> should be set from inside program
load "threatconeplot.txt"
#set arrow from 0,0 to 20,20 nohead lw 1 lc rgb "yellow"
#set arrow from 0,0 to -20,20 nohead lw 1 lc rgb "yellow"
plot 'carPosInVehFrame.dat' using 1:2 lt rgb "magenta" smooth bezier,
'./finalpathplotlines1.dat' using 1:2 lt rgb "green" with filledcurves closed,
'./finalpathplotlines.dat' using 1:2 lt rgb "orange" smooth bezier, './plotdata.dat' using
1:2 lt rgb "blue" with linespoints, './failedPathsplot.dat' using 1:2 lt rgb "cyan"
smooth Bezier
```

Gnuplot/simplot.plt

```
set terminal png font arial 14 size 800,600
set output filename
set multiplot layout 3, 1
set xlabel "Time(s)"
set grid
unset key
set ylabel "Throttle(%) and Brake(%)"
plot "./cardata.dat" using 1:4 lt rgb "green" with lines, " using 1:3 lt rgb "red" with
lines
set ylabel "Velocity(m/s)"
plot "./cardata.dat" using 1:9 lt rgb "blue" with lines, " using 1:10 lt rgb "green" with
lines
set ylabel "Steerangle(rad)"
plot "./cardata.dat" using 1:2 lt rgb "blue" with lines
unset multiplot
```

RRT/RRT\_class.h

```
/*
 * RRT_class.h
 *
 * Created on: Apr 16, 2014
 *      Author: Thomas
 */

#ifndef RRT_CLASS_H_
#define RRT_CLASS_H_
#define _USE_MATH_DEFINES

#define ENABLE_ABA           // Active brake assist
#define STORE_FAILED_PATHS   // Store failed paths

#include <cmath>
#include <vector>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <string.h>
#include <cstdlib>
#include <ctime>
using namespace std;

class RRT {
    typedef vector< vector<double> > ArrayList;
```

```

typedef vector<double> ArrayDbl;
typedef vector<int> ArrayInt;

enum _sensor_type { LIDAR, ULTASONIC };

class ABA {

    private:
        int brake_percent;
        bool ABA_event;
        double prev_distABA;
        double curr_distABA;
        double Vrel;
        double TTC;
        double target;
        bool updateFirstDistFlag;
        float GAIN;
        _sensor_type sensor_type;

    public:
        ABA(double, _sensor_type);
        void zero();
        void update(double, ArrayList, double);
        void update(double, double, double);
        bool getABAEvent() { return ABA_event; };
        int getABABrake() { return brake_percent; };
};

private:

    double Road_Lines[3];
    double Goal[2];
    double Temp_Node[3];
    int pathCount;

    double Path[50][5];
    double Final_Path[50][5];
    double Route_Tree[400][2];
    double Prev_Path[50][5];
    vector< vector< vector<double> > > failedPaths;

    void Get_Roadlines(ArrayDbl);
    bool roadDetected(ArrayDbl);
    void Get_Goal(ArrayDbl);
    void AddLinear(float,ArrayList,int);
    void AddRandom(float,ArrayList,int,ArrayDbl);

```

```

        double unifRandNumber();
        void Get_Path(ArrayList,float,ArrayDbl,float[]);
        bool CheckCollision(ArrayDbl, float[],ArrayDbl);
        bool enableABA;
        double minTimeToCollision;
        ABA *aba;

    public:
        RRT();
        ~RRT();
        float getRRTSteer();
        void Run_RRT(ArrayDbl,ArrayList,ArrayDbl);
        ArrayList getFinalPath();
        vector< vector< vector<double> > > getFailedPaths();
        bool getRRTVerify();
        bool getABAEvent() { return aba->getABAEvent(); };
        int getABABrake() { return aba->getABABrake(); };

    };
#endif /* RRT_CLASS_H_ */

```

### RRT/RRT\_class.cpp

```

/*
 * RRT_class.cpp
 *
 * Created on: Apr 16, 2014
 *      Author: Thomas
 */
//RRT_class.cpp
#include "RRT_class.h"
#include <bits/random.h>
random_device rd;
default_random_engine generator;
//mt19937 re(rd());
mt19937 re(time(0));
uniform_real_distribution<double> ud(0.0,1.0);
RRT::ABA::ABA(double targetFT, _sensor_type st = LIDAR) {
    target = targetFT;
    ABA_event = false;
    brake_percent = 0;
    prev_distABA = 0;
    curr_distABA = 0;
    Vrel = 0;
    TTC = 0;
    updateFirstDistFlag = true;
}

```

```

        sensor_type = st;
        GAIN = 1.25;
    }
    void RRT::ABA::zero() {
        ABA_event = false;
        brake_percent = 0;
    }
    // LiDAR based ABA update
    void RRT::ABA::update(double speed, ArrayList Obstacle_Array, double deltaT) {
        if(Obstacle_Array.size() > 0) {
            updateFirstDistFlag = true;
            for(int i=0; i < Obstacle_Array.size(); i++) {
                ArrayDbl obstacle = Obstacle_Array[i];
                if(((obstacle[0]+obstacle[2])/2) < -0.5)||((obstacle[0]-obstacle[2])/2) > 0.5))
                    ;
                else {
                    if(((obstacle[1]-obstacle[3])/2) < curr_distABA) ||
updateFirstDistFlag) {
                        curr_distABA = (obstacle[1]-obstacle[3])/2);
                        updateFirstDistFlag = false;
                    }
                }
            }
            if(prev_distABA!=0) {
                Vrel = (prev_distABA - curr_distABA)/deltaT;
            } else {
                Vrel = 0;
            }
            TTC = curr_distABA/(speed*3.2808399-Vrel);
            if(TTC < target && TTC > 0) {
                ABA_event = true;
                brake_percent = (int)((target-TTC)/target)*100*GAIN);
                if(brake_percent > 100)
                    brake_percent = 100;
            } else {
                brake_percent = 0;
                ABA_event = false;
            }
            // cout<<endl<<"ABA Event: "<<ABA_event<<endl<<"CurrDist:
            "<<curr_distABA<<endl<<"PrevDist: "<<prev_distABA<<endl<<"TTC:
            "<<TTC<<endl<<"Vrel: "<<Vrel<<endl<<"Brake %: "<<brake_percent<<endl;
            prev_distABA = curr_distABA;
        }
    }
}

```

```

// Ultrasonic based ABA update
void RRT::ABA::update(double speed, double distance, double deltaT) {
    if(distance > 0) {

        curr_distABA = distance;

        if(prev_distABA!=0) {
            Vrel = (prev_distABA - curr_distABA)/deltaT;
        } else {
            Vrel = 0;
        }
        TTC = curr_distABA/(speed*3.2808399-Vrel);
        if(TTC < target && TTC > 0) {
            ABA_event = true;
            brake_percent = (int)((target-TTC)/target)*100;
        } else {
            brake_percent = 0;
            ABA_event = false;
        }
        // cout<<endl<<"ABA Event: "<<ABA_event<<endl<<"CurrDist:
        "<<curr_distABA<<endl<<"PrevDist: "<<prev_distABA<<endl<<"TTC:
        "<<TTC<<endl<<"Vrel: "<<Vrel<<endl<<"Brake %: "<<brake_percent<<endl;

        prev_distABA = curr_distABA;
    }
}

RRT::RRT() {
    minTimeToCollision = 3.5;
    enableABA = true;
    if(enableABA)
        aba = new ABA(minTimeToCollision);
}

RRT::~RRT() {
    delete aba;
}

void RRT::Run_RRT(ArrayDbl Vehicle,ArrayList Obstacle_Array,ArrayDbl Road) {

    memset( Road_Lines,0,sizeof(Road_Lines) );
    memset( Temp_Node,0,sizeof(Temp_Node) );
    ArrayList Tree;
    memset( Route_Tree,0,sizeof(Route_Tree) );
    double Final_Tree[20][2];
    memset( Final_Tree,0,sizeof(Final_Tree) );
    float Node_Dist = 2.5f; // distance between nodes in tree
    bool Done = false;
    int Nodes = 1;
}

```

```

bool Rand_Go = false;
bool Fail = false;
int Iterations = 0;
float Vehicle_Geo_Actual[] = { 1.75f, 0.833f };
float Buffer = 0.15f;
float Vehicle_Geo[] = { Vehicle_Geo_Actual[0]+Buffer,
Vehicle_Geo_Actual[1]+Buffer };
memset( Final_Path, 0, sizeof(Final_Path) );

aba->zero();

if(Vehicle[3] == 0) {
    Vehicle[3] = 0.0000001;
}

failedPaths.clear();
//Num_Objects
int ObstacleNum = Obstacle_Array.size();

// No Road Detected
if( !roadDetected( Road ) )
    Fail = true;
// Road aligned along car axis
if(Road[2] == 0)
    Road[2] = 0.0000001;
// GET LINES OF ROAD BOUNDARIES
Get_Roadlines( Road );
// GET GOAL
Get_Goal( Road );
srand(time(NULL));
while((Done == false) && (Fail == false)) {

    bool NoGoGo = false;
    // ADD A TEMPORARY NODE TO [TREE]
    if(Tree.size() == 0) {
        ArrayDbl tempPush;
        tempPush.push_back( 0 ); tempPush.push_back( 0 );
tempPush.push_back( 0 );
        Tree.push_back( tempPush ); }
    if(Rand_Go == false) {
        AddLinear(Node_Dist, Tree, Nodes);
    }
    else {
        AddRandom(Node_Dist, Tree, Nodes, Road);
    }
    // CHECK TO SEE IF FINISHED
}

```

```

if((Temp_Node[0] == Goal[0])&&(Temp_Node[1] == Goal[1]))
    Done = true;
// GENERATE A PATH TO THE TEMPORARY NODE
Get_Path(Tree, Node_Dist, Vehicle, Vehicle_Geo );
// CHECK IF IT WAS A GOOD PATH
if(Path[0][0] > 9000) {
    NoGoGo = true;
}
else {
    // PATH IS GOOD, NOW CHECK FOR COLLISION
    // MULTIPLE OBJECT LOOP
    for(int i=0; i < ObstacleNum; i++) {
        ArrayDbl Obstacle = Obstacle_Array[i];
        NoGoGo = CheckCollision(Obstacle, Vehicle_Geo,
Road);
        if(NoGoGo)
            break;
    }
}
// ADD TEMPORARY NODE TO THE TREE IF IT IS VALID
if(NoGoGo == false) {
    Nodes = Nodes + 1;                                // INCREMENT
THE NODE COUNTER
    ArrayDbl tempAdd; tempAdd.push_back( Temp_Node[0] );
tempAdd.push_back( Temp_Node[1] ); tempAdd.push_back( Temp_Node[2] );
Tree.push_back(tempAdd);                                // ADDS THE
NODE TO [TREE]
    Rand_Go = false;                                // STOPS CHECKING
RANDOM NODES
    } else {
// TRY A NEW RANDOM NODE
    Rand_Go = true;
#ifndef STORE_FAILED_PATHS
        vector<double> steerandtimeValues;
        vector< vector<double> > retsteerandtimeValues;
        for(int i = 0; i < 50; i++) {
            steerandtimeValues.push_back(Path[i][0]);
            steerandtimeValues.push_back(Path[i][1]);
            steerandtimeValues.push_back(Path[i][2]);
            steerandtimeValues.push_back(Path[i][3]); //
push back steer
            steerandtimeValues.push_back(Path[i][4]); //
push back time
        }
        retsteerandtimeValues.push_back(steerandtimeValues);
        steerandtimeValues.clear();
    }
}

```

```

        }
        failedPaths.push_back(retsteerandtimeValues);
        retsteerandtimeValues.clear();
    #endif
}
if(Done) {
    copy(&Path[0][0], &Path[0][0]+50*5,&Final_Path[0][0]);
    int connectingNode = Nodes - 1; // INITIALIZES THE CONNECTION NODE FROM THE GOAL
    int finalNodes = 0;

    while(connectingNode >= 0) {
        ArrayDbl tempSet = Tree[connectingNode];
        Final_Tree[finalNodes][0] = tempSet[0];
        Final_Tree[finalNodes][1] = tempSet[1]; // MOVES THE PREVIOUS NODE IN
        THE TREE INTO THE FINAL TREE
        connectingNode = tempSet[2] - 1; // SETS THE NEXT CONNECTING NODE
        finalNodes = finalNodes + 1;
    }
    Iterations = Iterations + 1;
    if(Iterations > 200) {
        Fail = true;
    }
}
if(Fail) {
    Final_Path[0][0] = 9001;
    if(enableABA)
        aba->update(Vehicle[3], Obstacle_Array, 0.05);
}

// for(unsigned int i = 0; i < Tree.size(); i++) {
//     ArrayDbl tempPrint = Tree[i];
//     for( int j = 0; j < 3; j++) {
//         cout<<tempPrint[j]<<" ";
//     }
//     cout<<endl;
// }
// cout<<"Final Path"<<endl;
// for(int i = 0; i < 50; i++) {
//     // for(int j = 0; j < 5; j++) {
//         // cout<<Final_Path[i][j]<<" ";
//     }
//     // cout<<endl;
// }

```

```

}

vector< vector< vector<double> > > RRT::getFailedPaths() {
    return failedPaths;
}

bool RRT::roadDetected(ArrayDbl Road) {
    for(unsigned int i=0; i < Road.size();i++ ) {
        if(Road[i]!=0)
            return true;
    }
    return false;
}
float RRT::getRRTSteer() {
    return Final_Path[0][3];
}
bool RRT::getRRTVerify() {
    if(Final_Path[0][0] > 9000)
        return 0;
    else
        return 1;
}
vector< vector<double> > RRT::getFinalPath() {
    vector<double> steerandtimeValues;
    vector< vector<double> > retsteerandtimeValues;
    if(getRRTVerify()) { //&& not empty
        for(int i = 0; i < 50; i++) {
            steerandtimeValues.push_back(Final_Path[i][0]); // push back
x-position
            steerandtimeValues.push_back(Final_Path[i][1]); // push back
y-position
            steerandtimeValues.push_back(Final_Path[i][2]); // push back
yaw angle
            steerandtimeValues.push_back(Final_Path[i][3]); // push back
steer
            steerandtimeValues.push_back(Final_Path[i][4]); // push back
time
            retsteerandtimeValues.push_back(steerandtimeValues);
            steerandtimeValues.clear();
        }
        memcpy(Prev_Path, Final_Path, sizeof(Prev_Path));
    }
    else {
        // for(int i = 0; i < 50; i++) {
        //     steerandtimeValues.push_back(Path[i][0]);
    }
}

```

```

        // steerandtimeValues.push_back(Path[i][1]);
        // steerandtimeValues.push_back(Path[i][2]);
        // steerandtimeValues.push_back(Path[i][3]); // push back steer
        // steerandtimeValues.push_back(Path[i][4]); // push back time
        // retsteerandtimeValues.push_back(steerandtimeValues);
        // steerandtimeValues.clear();
    // }

    // failedPaths.push_back(retsteerandtimeValues);
    // retsteerandtimeValues.clear();

    // for(int i = 0; i < 50; i++) {
        // steerandtimeValues.push_back(Prev_Path[i][0]);
    // steerandtimeValues.push_back(Prev_Path[i][1]);
        // steerandtimeValues.push_back(Prev_Path[i][2]);
        // steerandtimeValues.push_back(Prev_Path[i][3]); // push back
    steer
        // steerandtimeValues.push_back(Prev_Path[i][4]); // push back
    time
        // retsteerandtimeValues.push_back(steerandtimeValues);
        // retsteerandtimeValues.clear();
        // steerandtimeValues.clear();
    // }

}
return retsteerandtimeValues;
}

void RRT::Get_Roadlines(ArrayDb1 Road) {
    double xl = Road[0];           // perpendicular distance from lidar to left
road boundary
    double xr = Road[1];           // perpendicular distance from lidar to
right road boundary
    double yaw = Road[2];           // angle between y-lidar coordinate and road
[degrees]
    // yaw is positive in the counterclockwise direction

    // RIGHT ROAD EQUATION y= m *x + br
    double br = (-xr)/tan( yaw*(M_PI/180.0) );           // y-intercept
    double m = -br/xr;           // slope

    // LEFT ROAD EQUATION y= m *x + bl
    double bl = (-xl)/tan( yaw*(M_PI/180.0) ); // y-intercept

    // ROADLINES=[m, bl, br] ;
    Road_Lines[0] = m;
    Road_Lines[1] = bl;
}

```

```

        Road_Lines[2] = br;
    }

void RRT::Get_Goal(ArrayDbl Road) {
    double xl = Road[0];           // perpendicular distance from lidar to left
road boundary
    double xr = Road[1];           // perpendicular distance from lidar to
right road boundary
    double yaw = Road[2];           // angle between y-lidar coordinate and
road [degrees]

    double m = Road_Lines[0];
    double Goal_Dist = 20.0;         // distance down the road the goal
is from the car
    double xrr= abs( cos( yaw*(M_PI/180.0) )*xr ); // closest distance to road
from car on right
    double xlr= abs( cos( yaw*(M_PI/180.0) )*xl ); // closest distance to road
from car on left
    int SIGN = yaw/abs(yaw);

    double Y;
    double x_dist;
    double w;

    // car is aligned straight with road
    if (yaw == 0) {
        Goal[0] = (xr+xl)/2;
        Goal[1] = Goal_Dist;
    }
    else {
        // car xl is the same as xr
        if (abs(xl)== xr)
            Y = cos( yaw*(M_PI/180.0) )*Goal_Dist;
        // car on the left side of road
        if (abs(xl) < xr) {
            x_dist = xrr - (xlr + xrr)/2;
            w = sin( yaw*( M_PI/180.0 ) ) * x_dist;
            Y = cos( yaw*(M_PI/180.0) ) * Goal_Dist + SIGN*w;
        }
        // car is on the right side of road
        if (abs(xl)> xr) {
            x_dist = xlr - (xlr + xrr)/2;
            w = sin( abs(yaw)*(M_PI/180.0) )*x_dist;
            Y = cos( yaw*(M_PI/180.0) ) * Goal_Dist + SIGN*w;
        }
        // now using a centerline of road to find x value
    }
}

```

```

        // if it is on the right side of road middle line will have - yint
        double Bl = Road_Lines[1]; // y-intercept of left road line
        double Br = Road_Lines[2]; // y-intercept of right road line
        double Bc = (Bl + Br)/2; // y-intercept of center road line

        // road center line
        double X = (Y - Bc)/m; // X value of goal
        Goal[0] = X;
        Goal[1] = Y;
    }
}

void RRT::AddLinear( float Node_Dist , ArrayList Tree , int Nodes ) {
    ArrayDbl Close_Node = Tree[Nodes-1]; // SETS NODE JUST ADDED TO
    THE CLOSEST NODE
        // TO TEST A STRAIGHT PATH FROM IT

        // DETERMINES THE DISTANCE TO THE GOAL
        double Goal_Dist = pow( pow( Goal[0] - Close_Node[0], 2 ) + pow( Goal[1] -
    Close_Node[1], 2 ), 0.5 );
        // CHECK TO SEE IF THE GOAL HAS BEEN REACHED BY
        COMPARING
        // THE DISTANCE TO THE GOAL TO THE NODE DISTANCE

        if(Goal_Dist <= Node_Dist) {
            Temp_Node[0] = Goal[0];
            Temp_Node[1] = Goal[1];
        }
        else {
            // LOCATION OF TEMPORARY NODE IN THE DIRECTION OF
            THE GOAL
            Temp_Node[0] = Close_Node[0] + (Node_Dist*(Goal[0] -
    Close_Node[0])) / Goal_Dist; // X
            Temp_Node[1] = Close_Node[1] + (Node_Dist*(Goal[1] -
    Close_Node[1])) / Goal_Dist; // Y
        }
        Temp_Node[2] = Nodes;
    }

void RRT::AddRandom( float Node_Dist , ArrayList Tree, int Nodes , ArrayDbl
Road ) {

    // CREATE RANDOM NODE ON ROAD
    // yaw is positive in the counterclockwise direction
}

```

```

        double xl = Road[0];           // perpendicular distance from lidar
to left road boundary
        double xr = Road[1];           // perpendicular distance from lidar
to right road boundary
        double yaw = Road[2];          // angle between y-lidar coordinate
and road
        double m = Road_Lines[0];       // slope of road

        double Ymin = 2.0; // (xr-xl)/2;           // minimum distance in y
direction to search in
        double Ymax = 30.0;            // maximum distance in y direction
to search in

//double X = (xr-xl)*unifRandNumber() + xl;           // RANDOM x-point
along line perpendicular to y-axis
//double randTemp = ud(generator);
        double X = (xr-xl)*ud(re) + xl;
//double Y = (Ymax-Ymin)*unifRandNumber() + Ymin;      // RANDOM y-
point along y-lidar axis
        double Y = (Ymax-Ymin)*ud(re) + Ymin;
        double Rand_Node[3];
        double Close_Node[2];
        double B_rand;

        if(yaw == 0)                  // the road is parallel to y-lidar axis
        Rand_Node[0] = X;
        Rand_Node[1] = Y;
    }
    else {                         // projects point along random line parallel
and within the road
        // Solving y = mx+b for y = 0 and x = Xrandom to find b
        B_rand = -m*X;             // random y-intercept based on X
        Rand_Node[0] = (Y - B_rand)/m; // RAND_NODE x value is then calulated
based on Y
        Rand_Node[1] = Y;           // RAND_NODE Y value is same as Y
    }

// SOLVING FOR THE DISTANCE TO THE RANDOM NODE
// INITIALIZES RAND_DIST TO A MAXIMUM VALUE WHICH WOULD
BE
// THE DISTANCE TO THE START OF THE TREE
        ArrayDbl temp = Tree[0];
        double Rand_Dist = pow(pow( Rand_Node[0] - temp[0], 2 ) + pow(
Rand_Node[1] - temp[1], 2 ), 0.5 );

// DETERMINING CLOSEST NODE TO THE RANDOM NODE

```

```

for(int i = 0; i < Nodes; i++) {

    // SOLVING FOR THE DISTANCE TO THE RANDOM NODE
    FROM THE A NODE IN [TREE]
    temp = Tree[i];
    double Temp_Dist = pow(pow( Rand_Node[0] - temp[0], 2 ) + pow(
    Rand_Node[1] - temp[1], 2 ), 0.5 );

    if (Temp_Dist <= Rand_Dist) {
        Rand_Dist = Temp_Dist;           // MAKES THE CHECKED
        DISTANCE THE NEW DISTANCE
        Close_Node[0] = temp[0];         // SETS THE CLOSEST
        NODE
        Close_Node[1] = temp[1];
        Temp_Node[2] = i + 1;           // MARKS WHAT NODE
        IT IS
    }
}

// LOCATION OF TEMPORARY NODE
Temp_Node[0] = Close_Node[0] + (Node_Dist*(Rand_Node[0] -
Close_Node[0]))/Rand_Dist; // X
Temp_Node[1] = Close_Node[1] + (Node_Dist*(Rand_Node[1] -
Close_Node[1]))/Rand_Dist; // Y

if (Temp_Node[1] < Close_Node[1]) {
    temp = Tree[ Temp_Node[2]-1 ];
    Temp_Node[2] = temp[2];
}
}

double RRT::unifRandNumber() {
    // Seed Random Number Generator
    //srand(time(0));
    double randomNum = ((double) rand() / ((double)RAND_MAX));
    return randomNum;
}

void RRT::Get_Path(ArrayList Tree,float Node_Dist,ArrayDbl Vehicle,float
Vehicle_Geo[]) {

    float angleLimit = 17.0;           // MAXIMUM ALLOWABLE ANGLE
    OF THE FRONT WHEEL
    int Route_Count = 0;             // NUMBER OF NODES IN THE ROUTE TO
    BE TESTED
    double center[] = { 0,0 };       // CENTER OF THE RADIUS OF
    CURVATURE
    double theta = 0;                // ANGLE OF THE CAR
    double phi = 0;                  // ARC ANGLE
}

```

```

double nu = 0;                                // LOOKAHEAD ANGLE
double LFW = 0;                                // LOOKAHEAD DISTANCE
double radius = 0;                             // RADIUS OF CURVATURE
int divLook = 8;                               // NUMBER OF DIVISIONS TO
LOOKAHEAD
    double DIV = 4;                            // NUMBER OF DIVISIONS PER
TREE SEGMENT
    double angleDiff = 4;                      // Differential step size along the arc
    int tests = 0;                            // NUMBER OF TESTS REQUIRED
    bool nogogo = false;                     // FLAG TO DETERMINE PATH FEASIBILITY
    float length = Vehicle_Geo[0];           // LENGTH OF THE VEHICLE
    double timeDiff = 0;                      // TIME BETWEEN NODES [s]
    double timeTot = 0;                      // TOTAL TIME FROM THE ORIGIN [s]

pathCount = 0;
memset( Path,0,sizeof(Path) );
// FRAMES

// WORK THE ROUTE BACKWARDS

// EXTRACT THE COMPLETED
PATH*****
    int connectingNode = (int)Temp_Node[2];// INITIALIZES THE
    CONNECTION NODE FROM THE TEMP NODE
    Route_Tree[0][0] = Temp_Node[0];
    Route_Tree[0][1] = Temp_Node[1];
    Route_Count = 0;

// Divide each branch of the tree into sub-steps
while(connectingNode > 0) {
    ArrayDbl temp = Tree[connectingNode-1];
    double xer = (1/DIV)*(Route_Tree[Route_Count][0] - temp[0]);
    double yer = (1/DIV)*(Route_Tree[Route_Count][1] - temp[1]);
    for(int i = 1; i <= DIV-1; i++) { // divides up each node length
        Route_Count = Route_Count + 1;
        Route_Tree[Route_Count][0] = Route_Tree[Route_Count -
1][0] - xer;
        Route_Tree[Route_Count][1] = Route_Tree[Route_Count -
1][1] - yer;
    }
    Route_Count = Route_Count + 1;
    // MOVES THE PREVIOUS NODE IN THE TREE INTO THE
FINAL TREE
    Route_Tree[Route_Count][0] = temp[0];
    Route_Tree[Route_Count][1] = temp[1];
    // SETS THE NEXT CONNECTING NODE
}

```

```

        connectingNode = temp[2];
    }

    // cout<<"Route Tree"<<endl;
    // for(int i = 0; i < 10; i++) {
        // cout<<Route_Tree[i][0]<< " "<<Route_Tree[i][1]<<endl;
    // }
    // cout<<endl;

    // Initialize Path and Counters
    int nn = 0;
    int cc = Route_Count;
    // Start at the Rout-Div element of the route
    tests = (Route_Count+1) - divLook;
    // Initial position of the vehicle based on LiDAR Coords
    Path[0][0] = 0;
    Path[0][1] = 0;
    Path[0][2] = 0;

    // For each point in ROUTE_TREE-DIV_LOOK, estimate a curve from the
    current
    // point to the DIV_LOOKth point and take a small step along that curve.
    for(int i = tests; i >= 1; --i) {
        // Make sure that the path is still valid
        if(nogogo == false) {
            // CALCULATE Lfw distance from current position to some
            node Div_Look ahead
            LFW = sqrt( pow( Path[nn][0] - Route_Tree[cc - divLook][0] ,
            2) + pow( Path[nn][1] - Route_Tree[cc-divLook][1] , 2 ) );
            // CALCULATE Nu The angle between current heading and the
            node that
            // is Div_look ahead Recall that Theta and Nu are negative in
            Quadrant 1 and
            // Positive in Quadrant 2.
            nu = -atan( ( Route_Tree[cc-divLook][0] - Path[nn][0] )/(
            Route_Tree[cc-divLook][1] - Path[nn][1] ) ) - Path[nn][2];
            // Check feasability of turn
            if( abs(nu) > M_PI/2 )
                nogogo = true;
            // Update counter for route tree
            cc = cc - 1;
            // if the node is directly ahead of the car and no turn is needed
            if(nu == 0) {
                radius = 0;
                // Update Path to send car directly to that point
                nn = nn + 1;
            }
        }
    }
}

```

```

Path[nn][0] = Route_Tree[cc][0];
Path[nn][1] = Route_Tree[cc][1];
// No change in theta if the car does not turn
Path[nn][2] = Path[nn-1][2];
Path[nn][3] = 0;
// Estimate time needed to complete maneuver
timeDiff = ( pow( pow( Path[nn][0] - Path[nn-1][0], 2) +
pow( Path[nn][1] - Path[nn-1][1], 2), 0.5 ) )/Vehicle[3];
timeTot = timeTot + timeDiff;
Path[nn][4] = timeTot;
//Path[nn][5] = radius;
}
else { // Turn required
// Get Current orientation of vehicle
theta = Path[nn][2];
// Radius of turn should be signed
radius = LFW/(2*sin(nu));
// Angle of inscribed chord
phi = 2*nu;
// Location of the center of turn
center[0] = Path[nn][0] - radius*cos(theta);
center[1] = Path[nn][1] - radius*sin(theta);
// Calculate steer angle needed to follow the given curve,
this
// approximation uses ackerman steer for a bicycle
model as an
// overly simplistic approximation of the steer angle,
future
// developments should add slip angle as well as
dynamic models
Path[nn][3] = atan(length/radius)*(180/M_PI);
// Check that steer angle is within physical limitations of
// hardware
if(abs(Path[nn][3]) > angleLimit)
nogogo = true;

// Next timestep
nn = nn + 1;
// add next node to tree along the arc [x,y,theta]
// if statement accounts for the case where the car is
turning
// right and the next node is to the left and vice versa
// Angle used to find x,y of next step
double alpha = (phi/angleDiff) + theta;
// find next x,y point
Path[nn][0] = center[0] + radius*cos(alpha);

```

```

Path[nn][1] = center[1] + radius*sin(alpha);
// Find heading at next step for perfect geometric
steering
    Path[nn][2] = (phi/angleDiff) + theta;
    // Check that the NN node is not behind the NN-1 Node
    if( Path[nn][1] <= Path[nn-1][1] )
        nogogo = true;

        // Estimate time needed to complete manuver
        timeDiff = abs( (phi/angleDiff)*(radius/Vehicle[3]) );
        timeTot = timeTot + timeDiff;
        Path[nn][4] = timeTot;
        pathCount = nn;
    }
}
// If any part of the path was not achieved, set the fail flag to
// try a new set of nodes
if( nogogo == true) {
    Path[0][0] = 9001;
    break;
}
}

bool RRT::CheckCollision(ArrayDbl Obstacle, float Vehicle_Geo[],ArrayDbl Road)
{
    int nn = 0;
    float length = Vehicle_Geo[0];
    float width = Vehicle_Geo[1];
    double V_ObstacleX = Obstacle[4];      // VELOCITY OF OBSTACLE in
X direction [ft/s]
    double V_ObstacleY = Obstacle[5];      // VELOCITY OF OBSTACLE in Y
direction [ft/s]
    bool NOGOGO = false;
    double ObstacleB[2][2] ;
    memset( ObstacleB,0,sizeof(ObstacleB) );

    double m = Road_Lines[0];                  // slope
    double Bl = Road_Lines[1];                  // left road boundary y-
intercept
    double Br = Road_Lines[2];                  // right road boundary y-
intercept
    double yaw = Road[2];

    // OBSTACLE
    double Center_Pos[] = { Obstacle[0], Obstacle[1] }; // [x,y]
    double O_HIT = Obstacle[3];      // HEIGHT[ft]
}

```

```

double O_WIT = Obstacle[2];      // WIDTH [ft]

// INITIAL POSITION
double OX1 = Center_Pos[0] - (O_WIT/2); // LEFT BOUNDARY
%THE OBSTACLE IS DEFINED BY A RECTANGLE
double OX2 = Center_Pos[0] + (O_WIT/2); // RIGHT BOUNDARY
    %THESE VALUES WILL BE COMING FROM SENSORS
double OY1 = Center_Pos[1] + (O_HIT/2); // UPPER BOUNDARY
%AND WILL BE SUBJECT TO CHANGE WHEN THE OBSTACLE MOVES
double OY2 = Center_Pos[1] - (O_HIT/2); // LOWER BOUNDARY

double Diag = sqrt( pow( Vehicle_Geo[0], 2 )+ pow( Vehicle_Geo[1], 2 ));

// THE CORNER TO CORNER LENGTH OF THE VEHICLE, CONSTANT

while ( (Path[nn+1][4] > 0) && (NOGOGO == false) && ((nn+1) < 50) ) {

    nn += 1;

    double x_0= Path[nn][0];                                // center x-
position of car
    double y_0= Path[nn][1];                                // center y-
position of car
    double theta = Path[nn][2]*(M_PI/180);                // angle of car
relative to Lidar-Coordinate system
    double Theta0 = atan((length/2)/(width/2)); // angle from center of car
to vertices

    // CALCULATE VERTICES OF THE CAR
    // VERTICES
    // 1=front left corner
    // 2=front right corner
    // 3=back left corner
    // 4=back right corner
    double Theta14 = Theta0 - theta; // Angle from vertices 1 and 4 from
lidar coordinate system
    double Theta23 = Theta0 + theta; // Angle from vertices 2 and 3 from
lidar coordinate system

    double dx14 = Diag/2*cos(Theta14); // Change in x-position from
parallel to coordinate system for 1 and 4
    double dx23 = Diag/2*cos(Theta23); // Change in x-position from
parallel to coordinate system for 2 and 3
    double dy14= Diag/2*sin(Theta14); // Change in y-position from
parallel to coordinate system for 1 and 4

```

```

        double dy23= Diag/2*sin(Theta23);      // Change in y-position from
parallel to coordinate system for 2 and 3

        double Vertices[4][2];
        Vertices[0][0] = x_0 - dx14;           // front left corner
        Vertices[0][1] = y_0 + dy14;
        Vertices[1][0] = x_0 - dx23;           // back left corner
        Vertices[1][1] = y_0 - dy23;
        Vertices[2][0] = x_0 + dx23;           // front right corner
        Vertices[2][1] = y_0 + dy23;
        Vertices[3][0] = x_0 + dx14;           // back right corner
        Vertices[3][1] = y_0 - dy14;

        // CHECK TO SEE IF VERTICES ARE OFF THE ROAD
        if(yaw == 0) {                         //
STRAIGHT ROAD SCENARIO
            for(int i=0; i < 4; i++) {
                if( (Vertices[i][0] < Road[0])||(Vertices[i][0] >
Road[1])) {
                    NOGOGO = true;
                    break;
                }
            }
        else {                                 // ANGLED
ROAD SCENARIOS
            double Y_max[4];
            double Y_min[4];
            // Finds the y values on the road based on the x-positions of the
vertices of the car
            if(yaw > 0) { // road veers left
                Y_max[0] = Vertices[0][0]*m + Bl;      Y_max[1] =
Y_max[2] = Vertices[2][0]*m + Bl;      Y_max[3] =
Y_min[0] = Vertices[0][0]*m + Br; Y_min[1] =
Y_min[2] = Vertices[2][0]*m + Br; Y_min[3] =
Y_max[0] = Vertices[0][0]*m + Br;      Y_max[1] =
Y_max[2] = Vertices[2][0]*m + Br;      Y_max[3] =
Y_min[0] = Vertices[0][0]*m + Bl; Y_min[1] =
Y_min[2] = Vertices[2][0]*m + Bl; Y_min[3] =
Y_max[0] = Vertices[0][0]*m + Bl;      Y_max[1] =
Y_max[2] = Vertices[2][0]*m + Bl;      Y_max[3] =
Y_min[0] = Vertices[0][0]*m + Bl; Y_min[1] =
Y_min[2] = Vertices[2][0]*m + Bl; Y_min[3] =
}
            if(yaw < 0) { //road veers right
                Y_max[0] = Vertices[0][0]*m + Br;      Y_max[1] =
Y_max[2] = Vertices[2][0]*m + Br;      Y_max[3] =
Y_min[0] = Vertices[0][0]*m + Bl; Y_min[1] =
Y_min[2] = Vertices[2][0]*m + Bl; Y_min[3] =
}
        }
    }
}

```

```

        // CHECK Y VALUES
        // makes sure that none of the y-values of the vertices of the car
are above or below the road
        for(int i=0; i < 4; i++) {
            if( (Y_max[i]-Vertices[i][1] < 0)|| (Vertices[i][1]-
Y_min[i]) < 0) {
                NOGOGO = true;
                break;
            }
        }
        // CHECK IF PATH AVOIDS OBSTACLE
        double Time_Tot = Path[nn][4];

        // SET THE OBSTACLE BOUNDING BOX
        ObstacleB[0][0] = OX1 + V_ObstacleX*Time_Tot; // LEFT SIDE X
        ObstacleB[0][1] = OY1 + V_ObstacleY*Time_Tot; // FURTHER

SIDE Y
X
Y

        // CHECK COLLISION WITH OBSTACLE
        double maxX =
fmax(fmax(fmax(Vertices[0][0], Vertices[1][0]), Vertices[2][0]), Vertices[3][0]);
        double minX =
fmin(fmin(fmin(Vertices[0][0], Vertices[1][0]), Vertices[2][0]), Vertices[3][0]);
        double maxY =
fmax(fmax(fmax(Vertices[0][1], Vertices[1][1]), Vertices[2][1]), Vertices[3][1]);
        double minY =
fmin(fmin(fmin(Vertices[0][1], Vertices[1][1]), Vertices[2][1]), Vertices[3][1]);
        if(NOGOGO == false) {
            if(minX > ObstacleB[1][0]) { // THE VEHICLE IS
ON THE RIGHT SIDE OF THE OBSTACLE

}

            else if(maxX < ObstacleB[0][0]) { // THE VEHICLE IS ON
THE LEFT SIDE OF THE OBSTACLE

}

            else if(maxY < ObstacleB[1][1]) { // THE VEHICLE IS
BEFORE THE OBSTACLE

}

```

```

        else if(minY > ObstacleB[0][1]) { // THE VEHICLE IS PAST
THE OBSTACLE

    }

else {
    NOGOGO = true;
}

}

return NOGOGO;
}

```

Segmentation/segmentation\_class.h

```

/*
 * segmentation_class.h
 *
 * Created on: Apr 16, 2014
 * Author: Thomas
 */

#ifndef SEGMENTATION_CLASS_H_
#define SEGMENTATION_CLASS_H_
#define _USE_MATH_DEFINES

#define ENABLE_LDW          // Lane-departure warning
#define ENABLE_LKA           // Lane-keep assist

#include <cmath>
#include <iostream>
#include <vector>
#include <algorithm>
#include <string.h>
using namespace std;

enum threat_type_t { NONE, OBSTACLE, ROADBOUNDARY,
RRT_MANUEVER };
enum threat_region_t { NA, LOW, MEDIUM, HIGH };

class Segmentation {
    typedef vector< vector<double> > ArrayList;
    typedef vector<int> ArrayInt;
    typedef vector<double> ArrayDbl;
private:
    ArrayList    AllObstacles;
    ArrayList    ReturnObstacles;
}

```

```

ArrayList    ReturnAllObstacles;
ArrayList    ThreateningObstacles;
ArrayDbl     ThreateningMarkers;
ArrayDbl     ReturnThreateningMarkers;
ArrayList    Obstacle;
ArrayList    RoadPts;
ArrayInt     PossibleRoadPts;
ArrayInt     toRemove;

threat_type_t threat_type;
threat_region_t threat_region;
int    Threshold;
double   maxNumPts;
double   ScalingFactor;
double   MaxVelocity;
double   angularRes_rad;
double   areamin;
int    noRoadCount;
bool   noRoadFlag;
bool   feasibleRoadway;
double roadBounds[6];
double   prevRoadBounds[6];
double   prevRoadSlope;
double   deltaYaw;
double   currentYaw;
double   prevYaw;
double   rotateIndices;
double   maxSlopeChange;
double   areaThreshold;
double   roadWidth;
double   areaBuffer;
double   coneSpacing;
double   spacingBuffer;
double   coneRoadBoundOffset;
double   threatThreshold;
double   distThreshold;
int    prevToggle;
int    Toggle;
double   minSpeed;
double   minDistance;
double threatConeAngle;
double   allowableDistToCone;
double   emergencyDistToCone;
double   thresholdManueverAngle;
double   maxStepSize;
int    BEAMNUMBER;

```

```

        bool roadBoundaryYawThreat_possible;

        double getBoxLength(double mini,double maxi,double mLength) {
            double dist=abs(maxi-mini)*ScalingFactor;
            if(dist<mLength)
                dist = mLength;
            return dist;
        };
        double chkMaxVelocity(double input,double accumlated,int
iterationNum) {
            if(std::abs(input)>MaxVelocity)
                return 0;
            else
                return input;
        };
        double* calculateSlopeIntercept(double[],double[]);
        bool checkRoadSlope(double);
        void segmentData();
        void determineThreat(double[]);
        void getObstacleInfo(double,double);
        void updateRoadBounds();
        void clusterData(double[],double[],double[],double[],double);
        void parseRoadBounds();
        void removeRoadCones();
        void cleanUpObstacles();
        void getCorrectObstacleInfo(double,double);
    public:
        Segmentation();
        Segmentation(int, int);
        Segmentation(int,
int,double,double,double,double,double,double,double,double,double,
double,double,double,double);
        Segmentation(int,
int,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,int);
        void
updateCorrectAdjustedVelocities(double[],double[],double[],double[],double);
        void identifyObstacles(double [],double[],double[],double[],double);
        void identifyObstacles(double[],double[],double[],double[],double,
double[],int);
        threat_type_t getThreatType();
        threat_region_t getThreatRegion();
        ArrayList getObstacleArray();
        ArrayList getAllObstacleArray();

```

```

        ArrayDbl returnRoadBounds() {    ArrayDbl returnBounds;
returnBounds.push_back(roadBounds[0]); returnBounds.push_back(roadBounds[1]);
returnBounds.push_back(roadBounds[2]);
                                return
returnBounds;
    }
    int returnThreat() { return Toggle; }
    double getThreatConeAngle() { return threatConeAngle; }
    double getThresholdDistance() { return distThreshold; }
    //~Segmentation();
};

#endif /* SEGMENTATION_CLASS_H_ */

```

Segmentation/segmentation\_class.cpp

```

/*
 * segmentation_class.cpp
 *
 * Created on: Apr 16, 2014
 *      Author: Thomas
 */
//segmentation_class.cpp
#include "segmentation_class.h"
template <typename T> int sgn(T val) {
    return (T(0) < val) - (val < T(0));
}
Segmentation::Segmentation() {
    Threshold = 12; maxNumPts = 75; MaxVelocity = 15.0; ScalingFactor = 1.1;
    BEAMNUMBER = 682;
    angularRes_rad = 0.36*(M_PI/180.0); areamin = 0.00001; noRoadCount = 0;
    noRoadFlag = false; prevRoadSlope = 0; deltaYaw = 0; currentYaw =
    0.0000001; prevYaw = 0.0000001;
    rotateIndices = 0; maxSlopeChange = 30; areaThreshold = 0.15; roadWidth =
    9.8;//12.3;
    areaBuffer = 0.125; coneSpacing = 4.0; spacingBuffer = 0.035;
    coneRoadBoundOffset = 1.0; maxStepSize = 1.0;
    threatThreshold = 1000065; distThreshold = 10; prevToggle = 0; Toggle = 0;
    minSpeed = -1.0;
    minDistance = 0.30; threatConeAngle = 45; allowableDistToCone = 3.0;
    thresholdManueverAngle = 10;
    emergencyDistToCone = 1.75;
    for(int i = 0;i < 6; i++) {
        roadBounds[i] = 0;
        prevRoadBounds[i] = 0;
    }
}

```

```

        feasibleRoadway = false;
    }
Segmentation::Segmentation(int Threshold, int maxNumPts, double ScalingFactor,
double roadWidth, double areaThreshold,
    double coneSpacing, double areaBuffer, double spacingBuffer, double
coneRoadBoundOffset, double threatThreshold, double distThreshold,
    double minSpeed, double minDistance, double threatConeAngle,
double allowableDistToCone, double thresholdManueverAngle, double
maxStepSize):
    Threshold(Threshold), maxNumPts(maxNumPts),
ScalingFactor(ScalingFactor), roadWidth(roadWidth),
areaThreshold(areaThreshold),
    coneSpacing(coneSpacing), areaBuffer(areaBuffer),
spacingBuffer(spacingBuffer), coneRoadBoundOffset(coneRoadBoundOffset),
threatThreshold(threatThreshold),
    distThreshold(distThreshold), minSpeed(minSpeed),
minDistance(minDistance), threatConeAngle(threatConeAngle),
allowableDistToCone(allowableDistToCone),
    thresholdManueverAngle(thresholdManueverAngle),
maxStepSize(maxStepSize)
{ BEAMNUMBER = 682; feasibleRoadway = false; }
Segmentation::Segmentation(int Threshold, int maxNumPts,double
ScalingFactor,double roadWidth,double areaThreshold,
    double coneSpacing,double areaBuffer,double spacingBuffer,double
coneRoadBoundOffset,double threatThreshold,double distThreshold,
    double minSpeed,double minDistance,double threatConeAngle,double
allowableDistToCone,double thresholdManueverAngle,double maxStepSize, int
BEAMNUMBER):
    Threshold(Threshold),maxNumPts(maxNumPts),ScalingFactor(ScalingFactor
),roadWidth(roadWidth),areaThreshold(areaThreshold),
    coneSpacing(coneSpacing),areaBuffer(areaBuffer),spacingBuffer(spacingBuf
fer),coneRoadBoundOffset(coneRoadBoundOffset),threatThreshold(threatThreshold
),
    distThreshold(distThreshold),minSpeed(minSpeed),minDistance(minDistance
),threatConeAngle(threatConeAngle),allowableDistToCone(allowableDistToCone),
    thresholdManueverAngle(thresholdManueverAngle),maxStepSize(maxStepSi
ze), BEAMNUMBER(BEAMNUMBER)
{ feasibleRoadway = false; }
// convert double[] to vectors and get rid of beamnumber
void Segmentation::identifyObstacles(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration) {
    // Cluster/Group Points within Point-Cloud

```

```

clusterData(x,y,prev_x,prev_y,scanDuration);
// Segment clustered data
segmentData();
// Parse/differentiate/Identify round bounds
parseRoadBounds();
// Update obstacles with true velocity after coordinate rotation determination
and reprocess obstacles if car has turned
updateRoadBounds();

// if(deltaYaw!=0) {
    // Sort data and clear buffers
    // AllObstacles.clear();
    // Obstacle.clear();
    // PossibleRoadPts.clear();
    // toRemove.clear();
    // RoadPts.clear();
    // updateCorrectAdjustedVelocities(x,y,prev_x,prev_y,scanDuration);
}
if(noRoadFlag==true || feasibleRoadway==false)
    Toggle = -1;
else if(Toggle!=-1)
    determineThreat( new double[2] { 0, 15 } );

// cout<< "All Obstacles"<<endl;
// for(unsigned int i=0;i<AllObstacles.size();i++) {
//     ArrayDbl temp = AllObstacles[i];
//     for(unsigned int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
//     cout<<endl;
// }
// cout<<endl;

// cout<<"Threatening Obstacles"<<endl;
// for(unsigned int i=0;i<ThreateningObstacles.size();i++) {
//     ArrayDbl temp = ThreateningObstacles[i];
//     for(unsigned int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
//     cout<<endl;
// }
// cout<<endl;

// ArrayDbl getRoadBounds = returnRoadBounds();
// for(unsigned int i=0; i < getRoadBounds.size(); i++) {
//     cout<<getRoadBounds[i]<<" ";
// }
// cout<<endl;
ReturnAllObstacles = AllObstacles;
ReturnObstacles = ThreateningObstacles;

```

```

ReturnThreateningMarkers = ThreateningMarkers;

AllObstacles.clear();
Obstacle.clear();
PossibleRoadPts.clear();
toRemove.clear();
RoadPts.clear();
ThreateningObstacles.clear();
ThreateningMarkers.clear();
}

void Segmentation::identifyObstacles(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration, double vehicleState[], int beamNumber) {
    BEAMNUMBER = beamNumber;
    // Cluster/Group Points within Point-Cloud
    clusterData(x,y,prev_x,prev_y,scanDuration);
    // Segment clustered data
    segmentData();
    // Parse/differentiate/Identify round bounds
    parseRoadBounds();
    // Update obstacles with true velocity after coordinate rotation determination
    and reprocess obstacles if car has turned
    updateRoadBounds();

    // if(deltaYaw!=0) {
        // Sort data and clear buffers
        // AllObstacles.clear();
        // Obstacle.clear();
        // PossibleRoadPts.clear();
        // toRemove.clear();
        // RoadPts.clear();
        // updateCorrectAdjustedVelocities(x,y,prev_x,prev_y,scanDuration);
    // }
    if(noRoadFlag==true || feasibleRoadway==false)
        Toggle = -1;
    else {
        determineThreat(vehicleState);
        // no wheel encoders
        // determineThreat( new double[2] { 0, 2 } );
    }

    // cout<< "All Obstacles"<<endl;
    // for(unsigned int i=0;i<AllObstacles.size();i++) {
    //     ArrayDbl temp = AllObstacles[i];
    //     for(unsigned int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
    //     cout<<endl;
    // }
}

```

```

// cout<<endl;
// cout<<"Threatening Obstacles"<<endl;
// for(int i=0;i<ThreateningObstacles.size();i++) {
    // ArrayDbl temp = ThreateningObstacles[i];
    // for(int j=0;j<temp.size();j++) { cout<<temp[j]<<" "; }
    // cout<<endl;
}
// ArrayDbl getRoadBounds = returnRoadBounds();
// for(unsigned int i=0; i < getRoadBounds.size(); i++) {
    cout<<getRoadBounds[i]<<" ";
}
// }

ReturnAllObstacles = AllObstacles;
ReturnObstacles = ThreateningObstacles;
ReturnThreateningMarkers = ThreateningMarkers;

AllObstacles.clear();
Obstacle.clear();
PossibleRoadPts.clear();
toRemove.clear();
RoadPts.clear();
ThreateningObstacles.clear();
ThreateningMarkers.clear();
}

vector< vector<double> > Segmentation::getObstacleArray() {
    return ReturnObstacles;
}
vector< vector<double> > Segmentation::getAllObstacleArray() {
    return ReturnAllObstacles;
}
void Segmentation::determineThreat(double vehicleState[]) {
    int count = 0;
    threat_type = NONE;
    threat_region = NA;
    double VehicleSpeed = sqrt(pow(vehicleState[0],2)+pow(vehicleState[1],2));
    if((VehicleSpeed >
minSpeed)&&(roadBounds[0]!=0)&&(roadBounds[1]!=0)) {
        for(unsigned int i = 0; i<AllObstacles.size();i++) {
            ArrayDbl temp = AllObstacles[i];
            double tempDist = sqrt(pow(temp[0],2)+pow(temp[1],2));
            if(tempDist>=minDistance) {
                double deltaTx = abs(temp[4]/tempDist);
                double deltaTy = abs(temp[5]/tempDist);
                double deltaT = min(deltaTx,deltaTy);
                double threat = 0;
                if(deltaT != 0)

```

```

        threat = 1/deltaT;

        if((threat>=threatThreshold)||tempDist<=distThreshold) {
            ArrayDbl threatAdd;
            threatAdd.push_back(temp[0]); threatAdd.push_back(temp[1]);
            threatAdd.push_back(temp[2]);
            threatAdd.push_back(temp[3]);
            //
            threatAdd.push_back(temp[4]+vehicleState[0]);
            //
            threatAdd.push_back(temp[5]+vehicleState[1]);
            threatAdd.push_back(temp[4]);
            threatAdd.push_back(temp[5]);
            // Check threat cone
            int insideLeftLine = sgn( temp[1] - tan(M_PI-
threatConeAngle*(M_PI/180))*temp[0] );
            int insideRightLine = sgn( temp[1] -
tan(threatConeAngle*(M_PI/180))*temp[0] );
            if( insideLeftLine>=0 && insideRightLine>=0
&& temp[1] > 0 ) { // Check if obstacle is inside threat cone
                Toggle = 1;

                ThreateningObstacles.push_back(threatAdd);
                threat_type = OBSTACLE;
            }
        }
    }

    if(Toggle<1) {
        double distToLeftBound =
abs(roadBounds[4])/sqrt(pow(roadBounds[3],2)+1);
        double distToRightBound =
abs(roadBounds[5])/sqrt(pow(roadBounds[3],2)+1);

        double smallestDistToBound;
        if(distToLeftBound < distToRightBound)
            smallestDistToBound = distToLeftBound;
        else
            smallestDistToBound = distToRightBound;

        if(smallestDistToBound <= emergencyDistToCone) {
            #ifdef ENABLE_LKA
            Toggle = 1;
            #endif
            threat_type = ROADBOUNDARY;
            threat_region = HIGH;
        }
    }
}

```

```

        } else
if((abs(currentYaw*180/M_PI)>=thresholdManueverAngle)&&(smallestDistToBoun
d<=allowableDistToCone)&&roadBoundaryYawThreat_possible) {
    #ifdef ENABLE_LKA
        Toggle = 1;
    #endif
    threat_type = ROADBOUNDARY;
    threat_region = MEDIUM;
} else
if((prevToggle==1)&&(abs(currentYaw*180/M_PI)>=thresholdManueverAngle)&&
roadBoundaryYawThreat_possible) {
    #ifdef ENABLE_LKA
        Toggle = 1;
    #endif
    threat_type = ROADBOUNDARY;
    threat_region = MEDIUM;
}
// Don't stop autonomous control in middle of autonomous manuever
//
if((Toggle==0)&&(prevToggle==1)&&(abs(currentYaw*180/M_PI)>=thresholdMan
ueverAngle)&&roadBoundaryYawThreat_possible)
    if((Toggle<1)&&(prevToggle>0)) {
        prevToggle = 0;
        Toggle = 1;
        threat_type = RRT_MANUEVER;
    }
    else {
        prevToggle = Toggle;
    }
}
void Segmentation::clusterData(double x[],double y[],double prev_x[],double
prev_y[],double scanDuration) {
    double previousVal = 0;
    for(int i=0; i<BEAMNUMBER; i++) {
        double mag = sqrt(pow(x[i],2)+pow(y[i],2));
        double step = 0;
        if(i==0) { previousVal = mag; }
        else { step = sqrt(pow(x[i-1]-x[i],2)+pow(y[i-1]-y[i],2)); }
        if(mag<=Threshold) {
            ArrayDbl temp;
            temp.push_back(x[i]); temp.push_back(y[i]);
            temp.push_back(prev_x[i]); temp.push_back(prev_y[i]);
            Obstacle.push_back(temp);
        }
    }
}

```

```

        if((Obstacle.size()>maxNumPts)||((step>maxStepSize))//&&(Obstacle.size()
>1))) // TODO: add parameter minNumPts (default = 2)? what about when size == 1
                getObstacleInfo(mag,scanDuration);
        }
        else {
            // If outside threshold and last point was inside threshold (and
            part of obstacle def.) end current obstacle def. or cluster
            if(previousVal<=Threshold) {
                getObstacleInfo(mag,scanDuration); }
            }
            previousVal = mag;
        }
    }

void Segmentation::getObstacleInfo(double distance,double deltaT) {
    int j;
    ArrayDbl temp;
    double xmin=0,xmax=0,ymin=0,ymax=0,VxPt=0,VyPt=0;
    int lastIt = Obstacle.size();
    if(Obstacle.size()>0) {
        for(j=0;j<lastIt;j++) {
            temp = Obstacle[j];
            if(j==0) { xmin=temp[0]; xmax=temp[0]; ymin=temp[1];
            ymax=temp[1]; VxPt=temp[2]; VyPt=temp[3]; }
            if(temp[0]<xmin)
                xmin = temp[0];
            if(temp[0]>xmax)
                xmax = temp[0];
            if(temp[1]<ymin)
                ymin = temp[1];
            if(temp[1]>ymax)
                ymax = temp[1];
            if((j!=0)&&(j!=lastIt)) {
                VxPt = VxPt+chkMaxVelocity((temp[2]-
temp[0])/deltaT, VxPt, j);
                VyPt = VyPt+chkMaxVelocity((temp[3]-
temp[1])/deltaT, VyPt, j);
            }
            temp.clear();
        }
        double xObstacle = (xmin+xmax)/2;
        double yObstacle = (ymin+ymax)/2;
        double dx = getBoxLength(xmin, xmax, distance*angularRes_rad);
        double dy = getBoxLength(ymin, ymax, distance*angularRes_rad);
        // Get obstacle length
    }
}

```

```

        double Vx = 0.0;//VxPt/Obstacle.size();
                           // Get obstacle x velocity
        double Vy = 0.0;//VyPt/Obstacle.size();
                           // Get obstacle y velocity
        double tempData[] = {xObstacle, yObstacle, dx, dy, Vx, Vy};
        vector<double> ObstacleData(tempData,
tempData+sizeof(tempData)/sizeof(double));
        AllObstacles.push_back(ObstacleData);
                           // Add obstacle data array to
array containing all obstacles
        Obstacle.clear();
    }
}

void Segmentation::segmentData() {
    for(unsigned int i=0;i<AllObstacles.size();i++) {
        ArrayDbl temp = AllObstacles[i];
        double tempArea = temp[2]*temp[3];
        if((tempArea<=areaThreshold)&&(tempArea>=areamin))
            PossibleRoadPts.push_back( i );
    }
}

void Segmentation::parseRoadBounds() {
    for(unsigned int i=0;i<PossibleRoadPts.size();i++) {
        int index1 = PossibleRoadPts[i];
        ArrayDbl compare1 = AllObstacles[index1];
        for(unsigned int j=0;j<PossibleRoadPts.size();j++) {
            if(j==i)
                continue;
            else {
                int index2 = PossibleRoadPts[j];
                ArrayDbl compare2 = AllObstacles[index2];
                double spacingDist = sqrt( pow(compare2[0]-
compare1[0],2)+pow(compare2[1]-compare1[1],2) );
                if((spacingDist>=(coneSpacing-
coneSpacing*spacingBuffer))&&(spacingDist<=(coneSpacing+coneSpacing*spacing
Buffer))) {
                    ArrayDbl temp;
                    temp.push_back(compare1[0]);
                    temp.push_back(compare1[1]);
                    RoadPts.push_back(temp);
                    temp.clear();
                    temp.push_back(compare2[0]);
                    temp.push_back(compare2[1]);
                    RoadPts.push_back(temp);
                    toRemove.push_back(index1);
                    toRemove.push_back(index2);
                }
            }
        }
    }
}

```

```

        }
    }
}

sort( toRemove.begin(), toRemove.end() );
toRemove.erase( unique( toRemove.begin(), toRemove.end() ),
toRemove.end() );
for(unsigned int i=0; i < RoadPts.size(); i++) {
    ArrayDbl compare1 = RoadPts[i];
    for(unsigned int j=0; j < RoadPts.size();j++) {
        if(j==i)
            continue;
        else {
            ArrayDbl compare2 = RoadPts[j];

            if((compare1[0]==compare2[0])&&(compare1[1]==compare2[1])) {
                RoadPts.erase(RoadPts.begin() + j);
                j=j-1;
            }
        }
    }
    removeRoadCones();
}
void Segmentation::removeRoadCones() {
    int remove = 0;
    for(unsigned int i=0; i < toRemove.size(); i++) {
        try {
            int index = toRemove[i];
            AllObstacles.erase(AllObstacles.begin() + (index - remove));
            remove++;
        } catch(...) { }
    }
}
void Segmentation::updateRoadBounds() {
    for(int i=0; i < 6; i++) {
        roadBounds[i] = 0;
    }
    double rightBound1[5] = { -1,0,0,0,0 };
    double rightBound2[5] = { -1,0,0,0,0 };
    double leftBound1[5] = { -1,0,0,0,0 };
    double leftBound2[5] = { -1,0,0,0,0 };
    int roadFlag = 0;

    for(unsigned int i=0; i < RoadPts.size(); i++) {
        ArrayDbl temp = RoadPts[i];

```

```

double tempDist = sqrt( pow( temp[0],2 ) + pow( temp[1],2 ) );
double theta = atan2( temp[1], temp[0] )*(180/M_PI);
if(temp[0] >= 0) {
    if(rightBound1[0] >= 0) {
        double spacingDist = sqrt( pow( rightBound1[3] -
temp[0],2 ) + pow(rightBound1[4] - temp[1],2 ) );
        if((spacingDist >
(coneSpacing+coneSpacing*spacingBuffer))&&(spacingDist<(coneSpacing-
coneSpacing*spacingBuffer))) {
            if(theta < rightBound1[1]) {
                rightBound1[0] = i;
                rightBound1[1] = theta;
                rightBound1[2] = tempDist;
                rightBound1[3] = temp[0];
                rightBound1[4] = temp[1];
            }
        }
    }
    else {
        rightBound2[0] = i;
        rightBound2[1] = theta;
        rightBound2[2] = tempDist;
        rightBound2[3] = temp[0];
        rightBound2[4] = temp[1];
    }
}
else {
    rightBound1[0] = i;
    rightBound1[1] = theta;
    rightBound1[2] = tempDist;
    rightBound1[3] = temp[0];
    rightBound1[4] = temp[1];
}
}
else {
    if(leftBound1[0] >= 0) {
        double spacingDist = sqrt( pow(leftBound1[3] -
temp[0],2 ) + pow(leftBound1[4] - temp[1],2 ) );
        if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
            if(theta > leftBound1[1]) {
                leftBound1[0] = i;
                leftBound1[1] = theta;
                leftBound1[2] = tempDist;
                leftBound1[3] = temp[0];
                leftBound1[4] = temp[1];
            }
        }
    }
}

```

```

        }
    }
    else {
        leftBound2[0] = i;
        leftBound2[1] = theta;
        leftBound2[2] = tempDist;
        leftBound2[3] = temp[0];
        leftBound2[4] = temp[1];
    }
}
else {
    leftBound1[0] = i;
    leftBound1[1] = theta;
    leftBound1[2] = tempDist;
    leftBound1[3] = temp[0];
    leftBound1[4] = temp[1];
}
}

if((rightBound1[0] >= 0)&&(rightBound2[0] >= 0)) {
    double spacingDist = sqrt( pow( rightBound1[3] -
rightBound2[3],2 ) + pow( rightBound1[4] - rightBound2[4],2 ) );
    if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
        if(rightBound2[1] > rightBound1[1]) {
            rightBound1[0] = rightBound2[0];
            rightBound1[1] = rightBound2[1];
            rightBound1[2] = rightBound2[2];
            rightBound1[3] = rightBound2[3];
            rightBound1[4] = rightBound2[4];
        }
    }
    else {
        roadFlag = 1;
        break;
    }
}
else if((leftBound1[0] >= 0)&&(leftBound2[0] >= 0)) {
    double spacingDist = sqrt( pow(leftBound1[3] -
leftBound2[3],2 ) + pow(leftBound1[4] - leftBound2[4],2) );
    if((spacingDist > (coneSpacing +
coneSpacing*spacingBuffer))&&(spacingDist < (coneSpacing -
coneSpacing*spacingBuffer))) {
        if(leftBound2[1] < leftBound1[1]) {
            leftBound1[0] = leftBound2[0];
            leftBound1[1] = leftBound2[1];
        }
    }
}

```

```

                leftBound1[2] = leftBound2[2];
                leftBound1[3] = leftBound2[3];
                leftBound1[4] = leftBound2[4];
            }
        }
    else {
        roadFlag = 2;
        break;
    }
}
}

double m = 0,d_l,d_r,yaw,offset = 0,b_r = 0,b_l = 0;
bool slopeCheck = true;
double *roadCalc;
roadBoundaryYawThreat_possible = false;
if(roadFlag == 1) {
    if(rightBound2[2] > rightBound1[2])
        roadCalc = calculateSlopeIntercept(rightBound1,rightBound2);
    else
        roadCalc = calculateSlopeIntercept(rightBound2,rightBound1);
    m = (*roadCalc); b_r = (*roadCalc+1));
    // if(prevRoadSlope != 0)
        // slopeCheck = checkRoadSlope();
    yaw = atan(1/m);
    if(yaw == 0) {           yaw = 0.0000001;   }
    slopeCheck = checkRoadSlope(yaw);

    if(slopeCheck == false) {

        double tempSlope = m;
        m = prevRoadSlope;
        prevRoadSlope = tempSlope;

        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {
            roadBounds[i] = 0;
        }
    } else {
        prevYaw = currentYaw;
        currentYaw = yaw;

        prevRoadSlope = m;
        offset = roadWidth/sin(yaw);
        d_r = -b_r/m;
        b_l = b_r + offset;
    }
}
}

```

```

d_l = -b_l/m;
roadBounds[0] = d_l+coneRoadBoundOffset;
roadBounds[1] = d_r-coneRoadBoundOffset;
roadBounds[2] = yaw*(180/M_PI);
roadBounds[3] = m;
roadBounds[4] = b_l;
roadBounds[5] = b_r;
memcpy(prevRoadBounds,roadBounds,6 *
sizeof(*roadBounds));

if(abs(d_l) < abs(d_r)) {
    if(b_l > 0) {
        roadBoundaryYawThreat_possible = true;
    }
} else {
    if(b_r > 0) {
        roadBoundaryYawThreat_possible = true;
    }
}
}

cleanUpObstacles();
delete[] roadCalc;

noRoadCount = 0;
noRoadFlag = false;
Toggle = 0;
}
else if(roadFlag == 2) {
    if(leftBound2[2] > leftBound1[2])
        roadCalc = calculateSlopeIntercept(leftBound1,leftBound2);
    else
        roadCalc = calculateSlopeIntercept(leftBound2,leftBound1);
    m = (*roadCalc); b_l = (*(roadCalc+1));
    yaw = atan(1/m);
    if(yaw == 0) { yaw = 0.0000001; }
    slopeCheck = checkRoadSlope(yaw);

    if(slopeCheck == false) {

        double tempSlope = m;
        m = prevRoadSlope;
        prevRoadSlope = tempSlope;

        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {

```

```

                roadBounds[i] = 0;
            }
        } else {
            prevYaw = currentYaw;
            currentYaw = yaw;

            prevRoadSlope = m;
            offset = roadWidth/sin(yaw);
            d_l = -b_l/m;
            b_r = b_l - offset;
            d_r = -b_r/m;
            roadBounds[0] = d_l+coneRoadBoundOffset;
            roadBounds[1] = d_r-coneRoadBoundOffset;
            roadBounds[2] = yaw*(180/M_PI);
            roadBounds[3] = m;
            roadBounds[4] = b_l;
            roadBounds[5] = b_r;
            memcpy(prevRoadBounds,roadBounds,6 *
sizeof(*roadBounds));

            if(abs(d_l) < abs(d_r)) {
                if(b_l > 0) {
                    roadBoundaryYawThreat_possible = true;
                }
            } else {
                if(b_r > 0) {
                    roadBoundaryYawThreat_possible = true;
                }
            }
        }

        cleanUpObstacles();
        delete[] roadCalc;

        noRoadCount = 0;
        noRoadFlag = false;
        Toggle = 0;
    }
    else {
        if((rightBound1[0] <
0)&&(rightBound2[0]<0)&&(leftBound1[0]<0)&&(leftBound2[0]<0))
            noRoadCount++;
        if(noRoadCount>3) {
            noRoadFlag = true;
            feasibleRoadway = false;
            for(int i=0; i < 6; i++) {

```

```

                roadBounds[i] = 0;
            }
        } else {
            memcpy(roadBounds,prevRoadBounds,6 *
sizeof(*prevRoadBounds));
            currentYaw = atan(1/prevRoadSlope);
            feasibleRoadway = true;
            roadBoundaryYawThreat_possible = false;
        }
    }
}

void Segmentation::cleanUpObstacles() {
    double br = -roadBounds[1]/tan(roadBounds[2]*(M_PI/180));
    double m = -br/roadBounds[1];
    double bl = -roadBounds[0]/tan(roadBounds[2]*(M_PI/180));

    double rx1 = -100;
    double ry1 = m*-100+br;
    double rx2 = 100;
    double ry2 = m*100+br;

    double lx1 = -100;
    double ly1 = m*-100+bl;
    double lx2 = 100;
    double ly2 = m*100+bl;

    for(unsigned int i=0;i < AllObstacles.size(); i++) {
        ArrayDbl temp = AllObstacles[i];
        int leftofRightRoadline = sgn( -(rx2-rx1)*(temp[1]-ry1) - (ry2-
ry1)*(temp[0]-rx1) );
        int rightofLeftRoadline = sgn( (lx2-lx1)*(temp[1]-ly1) - (ly2-
ly1)*(temp[0]-lx1) );
        if(leftofRightRoadline != rightofLeftRoadline) {
            AllObstacles.erase(AllObstacles.begin() + i);
            i = i-1;
        }
    }

    // Check for feasible roadway (car within road boundaries)
    int leftofRightRoadline = sgn( -(rx2-rx1)*(-ry1) - (ry2-ry1)*(-rx1) );
    int rightofLeftRoadline = sgn( (lx2-lx1)*(-ly1) - (ly2-ly1)*(-lx1) );

    if(leftofRightRoadline != rightofLeftRoadline) {
        feasibleRoadway = false;
        for(int i=0; i < 6; i++) {
            roadBounds[i] = 0;
    }
}

```

```

        }
    } else {
        feasibleRoadway = true;
    }
}
bool Segmentation::checkRoadSlope(double yaw) {
    // deltaYaw = currentYaw - prevYaw;
    deltaYaw = yaw - prevYaw;
    if(abs(deltaYaw)*(180/M_PI) >= maxSlopeChange) {
        rotateIndices = 0;
        return false;
    }
    else {
        rotateIndices = -(deltaYaw/angularRes_rad);
        return true;
    }
}
threat_type_t Segmentation::getThreatType() {
    return threat_type;
}
threat_region_t Segmentation::getThreatRegion() {
    return threat_region;
}
double* Segmentation::calculateSlopeIntercept(double closeBound[],double farBound[]) {
    double slope = 0;
    double intercept = 0;
    slope = ((farBound[4] - closeBound[4])/(farBound[3] - closeBound[3]));
    intercept = (closeBound[4] - slope*closeBound[3]);
    double *retArray = new double[2];
    *retArray = slope;
    *(retArray+1) = intercept;

    return retArray;
}

```

## **Appendix F              Development Board Specification Matrix**

*Price:* This comparison metric was selected to take into account the varying cost between SBCs and was given a small weighting influence as no board is prohibitively expensive. In fact, most are rather inexpensive.

*Processor:* General-purpose processors are often used in embedded systems and present many benefits. They are programmable digital systems intended to solve computation tasks in a diverse array of applications and they largely influence the features, functionality, and performance of SBCs; processor consideration and selection is of key importance to the design of an embedded system.

*Processor Speed & Benchmark Performance:* This comparison metric was selected to take into account the varying processor speeds between SBCs and was given a moderate weighting influence. Processor speed is a difficult aspect to measure and compare as it can greatly vary between SBCs. The more useful comparison metric is to compare the throughput often estimated as the number of instructions per clock cycle per second. This can still vary across processors as one processor may require more instructions to perform the same computation. However, a benchmark program intends to be run on different processors in order to compare their performance. One such benchmarking program is the Dhrystone Benchmark that was written in 1984 by Reinhold Weicker in C and is available in the public domain. Despite the use of benchmarking programs, the validity of benchmark data is a subject of great controversy, especially in regards to measuring embedded processor performance. Different benchmarking standards provide metrics of the processor's performance. The Whetstone standard is a synthetic benchmark for evaluating the performance of computers that measures the Millions of Whetstone Instructions per Second (MWIPS). The Dhrystone benchmark contains no floating point operations while the Whetstone benchmark does. Often single-board computers and their processors advertise their speed by posting a measure of the Million Instructions per Second (MIPS); however, the particular benchmarking program used may not be listed.

*RAM:* This comparison metric was selected to account for the varying RAM between SBCs and was given a moderately large weighting influence as each SBC must meet the minimum estimated RAM. In addition, more RAM means better performance and more opportunities for future development. In order to estimate the required minimum processing throughput, RAM and onboard storage, the existing code was examined to evaluate how much memory was necessary to hold all variables for a single iteration. Each variable's memory footprint was considered at its maximum possible size and all summed together with some contingencies to estimate the maximum required RAM.

*Matlab/Simulink Support:* This comparison metric was selected to differentiate SBCs with explicit optimized embedded coder support for Matlab/Simulink programs. As the software runs in Simulink (which these SBCs are not capable of running), it must be ported/re-written into a compatible language these boards can interpret. Matlab embedded coder is a toolbox that allows Matlab/Simulink programs to be seamlessly compiled into C/C++ libraries and executables. This metric was given a relatively small weighting parameter as even the SBCs that are not explicitly supported may still run the code generated with this toolbox.

*Community Support Network & Learning Curve:* This comparison metric was selected to quantify the difference between the size and quality of the support network between each SBC. Only development boards that have been embraced by the open-source community were considered because this often coincides with SBCs that are relatively inexpensive, widely available, and well-documented.

*Peripherals:* This comparison metric was selected to account for the different number and types of peripherals each SBC possesses. Peripherals of particular interest for this project are UART, I<sup>2</sup>C, SPI, USB, and GPIO. These on-chip peripherals allow for interfacing to hardware component such as the LiDAR sensor, the user control station, the sensors, and the motors. Any additional peripherals will allow for future development and/or communication flexibility.

*Power Consumption:* Embedded computing systems all have various power requirements such as current draw and operating voltage range. This comparison metric takes into account the operating voltage range and current draw of the considered SBCs with the available voltage rails.

Requirement ID	Requirement Name	Description	Priority	Status	Last Update	Owner	Properties												Comments
							Label	Value	Min	Max	Unit	Memory (GB)	Processor (MHz)	Ledging Line	Interconnection (min. layer)	Type/Model	Series	Color	Serial Number
REQ-001	Initial System Configuration	Initial System Configuration	High	Planned	2023-01-01	John Doe	Label1	Value1	1	100	Unit1	8	1600	1000	Layer1	Standard	Black	1234567890	Initial System Configuration
REQ-002	Network Expansion Phase 1	Network Expansion Phase 1	Medium	In Progress	2023-02-15	Jane Smith	Label2	Value2	10	100	Unit2	16	2000	1200	Layer2	Advanced	Red	9876543210	The Network Expansion Phase 1 is currently in progress. All new nodes have been added and are now fully operational.
REQ-003	Cloud Migration Project	Cloud Migration Project	High	Planned	2023-03-31	Bob Johnson	Label3	Value3	20	100	Unit3	32	2500	1500	Layer3	Proven	Blue	5432109876	The Cloud Migration Project has been scheduled for March 31st. All resources are currently being prepared.
REQ-004	Security Audit and Compliance	Security Audit and Compliance	Medium	In Progress	2023-04-15	Mike Williams	Label4	Value4	30	100	Unit4	64	3000	1800	Layer4	Enhanced	Green	1234567890	The Security Audit and Compliance project is currently in progress. All findings are being addressed.
REQ-005	Performance Optimization	Performance Optimization	High	Planned	2023-05-31	Sarah Lee	Label5	Value5	40	100	Unit5	128	3500	2000	Layer5	Optimized	Yellow	9876543210	The Performance Optimization project has been planned for May 31st. All performance metrics are currently being analyzed.
REQ-006	Data Center Upgrade	Data Center Upgrade	Medium	In Progress	2023-06-30	David Green	Label6	Value6	50	100	Unit6	256	4000	2500	Layer6	Upgraded	Grey	5432109876	The Data Center Upgrade project is currently in progress. All hardware has been installed and is now operational.
REQ-007	AI Integration Phase 1	AI Integration Phase 1	High	Planned	2023-07-31	Emily White	Label7	Value7	60	100	Unit7	512	4500	3000	Layer7	AI Enabled	Purple	1234567890	The AI Integration Phase 1 project has been planned for July 31st. All AI components are currently being integrated.
REQ-008	System Monitoring and Logging	System Monitoring and Logging	Medium	In Progress	2023-08-15	Frank Black	Label8	Value8	70	100	Unit8	1024	5000	3500	Layer8	Monitored	Orange	9876543210	The System Monitoring and Logging project is currently in progress. All logs are being collected and analyzed.
REQ-009	Resource Allocation and Scheduling	Resource Allocation and Scheduling	High	Planned	2023-09-30	Grace Grey	Label9	Value9	80	100	Unit9	2048	5500	4000	Layer9	Allocated	White	5432109876	The Resource Allocation and Scheduling project has been planned for September 30th. All resources are currently being allocated.
REQ-010	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2023-10-15	Hannah Blue	Label10	Value10	90	100	Unit10	4096	6000	4500	Layer10	Redundant	Red	1234567890	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-011	Compliance and Legal Requirements	Compliance and Legal Requirements	High	Planned	2023-11-30	Ivan Red	Label11	Value11	100	100	Unit11	8192	6500	5000	Layer11	Compliant	Cyan	9876543210	The Compliance and Legal Requirements project has been planned for November 30th. All legal requirements are currently being reviewed.
REQ-012	System Health and Monitoring	System Health and Monitoring	Medium	In Progress	2023-12-15	Jessica Yellow	Label12	Value12	110	100	Unit12	16384	7000	5500	Layer12	Monitored	Yellow	5432109876	The System Health and Monitoring project is currently in progress. All monitoring tools are being used to track system health.
REQ-013	System Performance Tuning	System Performance Tuning	High	Planned	2024-01-31	Karen Purple	Label13	Value13	120	100	Unit13	32768	7500	6000	Layer13	Tuned	White	1234567890	The System Performance Tuning project has been planned for January 31st. All performance tuning steps are currently being implemented.
REQ-014	System Security Audit	System Security Audit	Medium	In Progress	2024-02-28	Liam Green	Label14	Value14	130	100	Unit14	65536	8000	6500	Layer14	Audited	Orange	9876543210	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-015	System Resource Management	System Resource Management	High	Planned	2024-03-31	Mia Blue	Label15	Value15	140	100	Unit15	131072	8500	7000	Layer15	Managed	Red	5432109876	The System Resource Management project has been planned for March 31st. All resource management strategies are currently being implemented.
REQ-016	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2024-04-30	Noah Red	Label16	Value16	150	100	Unit16	262144	9000	7500	Layer16	Redundant	Red	1234567890	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-017	System Performance Tuning	System Performance Tuning	High	Planned	2024-05-31	Olivia Yellow	Label17	Value17	160	100	Unit17	512384	9500	8000	Layer17	Tuned	White	9876543210	The System Performance Tuning project has been planned for May 31st. All performance tuning steps are currently being implemented.
REQ-018	System Security Audit	System Security Audit	Medium	In Progress	2024-06-30	Parker Purple	Label18	Value18	170	100	Unit18	1024768	10000	8500	Layer18	Audited	Yellow	5432109876	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-019	System Resource Management	System Resource Management	High	Planned	2024-07-31	Riley Green	Label19	Value19	180	100	Unit19	2049120	10500	9000	Layer19	Managed	Red	1234567890	The System Resource Management project has been planned for July 31st. All resource management strategies are currently being implemented.
REQ-020	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2024-08-31	Sophie Blue	Label20	Value20	190	100	Unit20	4098240	11000	9500	Layer20	Redundant	Red	9876543210	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-021	System Performance Tuning	System Performance Tuning	High	Planned	2024-09-30	Ulysses Red	Label21	Value21	200	100	Unit21	8196480	11500	10000	Layer21	Tuned	White	5432109876	The System Performance Tuning project has been planned for September 30th. All performance tuning steps are currently being implemented.
REQ-022	System Security Audit	System Security Audit	Medium	In Progress	2024-10-31	Vivian Yellow	Label22	Value22	210	100	Unit22	16392960	12000	10500	Layer22	Audited	Orange	1234567890	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-023	System Resource Management	System Resource Management	High	Planned	2024-11-30	Wade Purple	Label23	Value23	220	100	Unit23	32785920	12500	11000	Layer23	Managed	Red	9876543210	The System Resource Management project has been planned for November 30th. All resource management strategies are currently being implemented.
REQ-024	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2024-12-31	Xavier Green	Label24	Value24	230	100	Unit24	65571840	13000	11500	Layer24	Redundant	Red	5432109876	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-025	System Performance Tuning	System Performance Tuning	High	Planned	2025-01-31	Yara Blue	Label25	Value25	240	100	Unit25	13114360	13500	12000	Layer25	Tuned	White	1234567890	The System Performance Tuning project has been planned for January 31st. All performance tuning steps are currently being implemented.
REQ-026	System Security Audit	System Security Audit	Medium	In Progress	2025-02-28	Zoey Red	Label26	Value26	250	100	Unit26	26228720	14000	12500	Layer26	Audited	Yellow	9876543210	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-027	System Resource Management	System Resource Management	High	Planned	2025-03-31	Abigail Yellow	Label27	Value27	260	100	Unit26	51457440	14500	13000	Layer27	Managed	Red	5432109876	The System Resource Management project has been planned for March 31st. All resource management strategies are currently being implemented.
REQ-028	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2025-04-30	Bryce Purple	Label28	Value28	270	100	Unit28	102914880	15000	13500	Layer28	Redundant	Red	1234567890	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-029	System Performance Tuning	System Performance Tuning	High	Planned	2025-05-31	Caitlyn Green	Label29	Value29	280	100	Unit29	205829760	15500	14000	Layer29	Tuned	White	9876543210	The System Performance Tuning project has been planned for May 31st. All performance tuning steps are currently being implemented.
REQ-030	System Security Audit	System Security Audit	Medium	In Progress	2025-06-30	Damon Red	Label30	Value30	290	100	Unit30	411659520	16000	14500	Layer30	Audited	Yellow	5432109876	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-031	System Resource Management	System Resource Management	High	Planned	2025-07-31	Ella Yellow	Label31	Value31	300	100	Unit31	823319040	16500	15000	Layer31	Managed	Red	1234567890	The System Resource Management project has been planned for July 31st. All resource management strategies are currently being implemented.
REQ-032	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2025-08-31	Fiona Purple	Label32	Value32	310	100	Unit32	1646638080	17000	15500	Layer32	Redundant	Red	9876543210	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-033	System Performance Tuning	System Performance Tuning	High	Planned	2025-09-30	Gavin Green	Label33	Value33	320	100	Unit33	3293276160	17500	16000	Layer33	Tuned	White	5432109876	The System Performance Tuning project has been planned for September 30th. All performance tuning steps are currently being implemented.
REQ-034	System Security Audit	System Security Audit	Medium	In Progress	2025-10-31	Hannah Red	Label34	Value34	330	100	Unit34	6586552320	18000	16500	Layer34	Audited	Yellow	1234567890	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-035	System Resource Management	System Resource Management	High	Planned	2025-11-30	Ivan Yellow	Label35	Value35	340	100	Unit35	13173104640	18500	17000	Layer35	Managed	Red	9876543210	The System Resource Management project has been planned for November 30th. All resource management strategies are currently being implemented.
REQ-036	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2025-12-31	Jessica Purple	Label36	Value36	350	100	Unit36	26346209280	19000	17500	Layer36	Redundant	Red	5432109876	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-037	System Performance Tuning	System Performance Tuning	High	Planned	2026-01-31	Karen Green	Label37	Value37	360	100	Unit37	52692418560	19500	18000	Layer37	Tuned	White	1234567890	The System Performance Tuning project has been planned for January 31st. All performance tuning steps are currently being implemented.
REQ-038	System Security Audit	System Security Audit	Medium	In Progress	2026-02-28	Liam Red	Label38	Value38	370	100	Unit38	105384837120	20000	18500	Layer38	Audited	Yellow	9876543210	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-039	System Resource Management	System Resource Management	High	Planned	2026-03-31	Mia Yellow	Label39	Value39	380	100	Unit39	210769674240	20500	19000	Layer39	Managed	Red	5432109876	The System Resource Management project has been planned for March 31st. All resource management strategies are currently being implemented.
REQ-040	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2026-04-30	Noah Purple	Label40	Value40	390	100	Unit40	421539348480	21000	19500	Layer40	Redundant	Red	1234567890	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-041	System Performance Tuning	System Performance Tuning	High	Planned	2026-05-31	Olivia Green	Label41	Value41	400	100	Unit41	843078696960	21500	20000	Layer41	Tuned	White	9876543210	The System Performance Tuning project has been planned for May 31st. All performance tuning steps are currently being implemented.
REQ-042	System Security Audit	System Security Audit	Medium	In Progress	2026-06-30	Parker Red	Label42	Value42	410	100	Unit42	1686157393920	22000	20500	Layer42	Audited	Yellow	5432109876	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-043	System Resource Management	System Resource Management	High	Planned	2026-07-31	Riley Yellow	Label43	Value43	420	100	Unit43	3372314787840	22500	21000	Layer43	Managed	Red	1234567890	The System Resource Management project has been planned for July 31st. All resource management strategies are currently being implemented.
REQ-044	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2026-08-31	Sophie Purple	Label44	Value44	430	100	Unit44	6744629575680	23000	21500	Layer44	Redundant	Red	9876543210	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-045	System Performance Tuning	System Performance Tuning	High	Planned	2026-09-30	Ulysses Green	Label45	Value45	440	100	Unit45	13489259151360	23500	22000	Layer45	Tuned	White	5432109876	The System Performance Tuning project has been planned for September 30th. All performance tuning steps are currently being implemented.
REQ-046	System Security Audit	System Security Audit	Medium	In Progress	2026-10-31	Vivian Red	Label46	Value46	450	100	Unit46	26978518302720	24000	22500	Layer46	Audited	Yellow	1234567890	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-047	System Resource Management	System Resource Management	High	Planned	2026-11-30	Wade Yellow	Label47	Value47	460	100	Unit47	53957036605440	24500	23000	Layer47	Managed	Red	9876543210	The System Resource Management project has been planned for November 30th. All resource management strategies are currently being implemented.
REQ-048	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2026-12-31	Xavier Purple	Label48	Value48	470	100	Unit48	107914073210880	25000	23500	Layer48	Redundant	Red	5432109876	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-049	System Performance Tuning	System Performance Tuning	High	Planned	2027-01-31	Yara Green	Label49	Value49	480	100	Unit49	215828146421760	25500	24000	Layer49	Tuned	White	1234567890	The System Performance Tuning project has been planned for January 31st. All performance tuning steps are currently being implemented.
REQ-050	System Security Audit	System Security Audit	Medium	In Progress	2027-02-28	Zoey Red	Label50	Value50	490	100	Unit50	431656292843520	26000	24500	Layer50	Audited	Yellow	9876543210	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-051	System Resource Management	System Resource Management	High	Planned	2027-03-31	Abigail Yellow	Label51	Value51	500	100	Unit51	863312585687040	26500	25000	Layer51	Managed	Red	5432109876	The System Resource Management project has been planned for March 31st. All resource management strategies are currently being implemented.
REQ-052	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2027-04-30	Bryce Purple	Label52	Value52	510	100	Unit52	172662517137440	27000	25500	Layer52	Redundant	Red	1234567890	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-053	System Performance Tuning	System Performance Tuning	High	Planned	2027-05-31	Caitlyn Green	Label53	Value53	520	100	Unit53	345325034274880	27500	26000	Layer53	Tuned	White	9876543210	The System Performance Tuning project has been planned for May 31st. All performance tuning steps are currently being implemented.
REQ-054	System Security Audit	System Security Audit	Medium	In Progress	2027-06-30	Damon Red	Label54	Value54	530	100	Unit54	690650068549760	28000	26500	Layer54	Audited	Yellow	5432109876	The System Security Audit project is currently in progress. All security vulnerabilities are being identified and addressed.
REQ-055	System Resource Management	System Resource Management	High	Planned	2027-07-31	Ella Yellow	Label55	Value55	540	100	Unit55	1381300137099520	28500	27000	Layer55	Managed	Red	1234567890	The System Resource Management project has been planned for July 31st. All resource management strategies are currently being implemented.
REQ-056	System Redundancy and Failover	System Redundancy and Failover	Medium	In Progress	2027-08-31	Fiona Purple	Label56	Value56	550	100	Unit56	2762600274199040	29000	27500	Layer56	Redundant	Red	9876543210	The System Redundancy and Failover project is currently in progress. All failover paths are being tested.
REQ-057	System Performance Tuning	System Performance Tuning	High	Planned	2027-09-30	Gavin Green	Label57	Value57	560	100	Unit57	552520054839840	29500	28000	Layer57	Tuned	White	5432109876	The System Performance Tuning project has been planned for September 30th. All performance tuning steps are currently being implemented.
REQ-058	System Security Audit	System Security Audit	Medium	In Progress	2027-10-31	Hannah Red	Label58	Value58	570	100	Unit58	11050							