THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL AND NUCLEAR ENGINEERING

PREPARATION OF LIDAR POINT CLOUD DATA FOR USE IN A ROS/GAZEBO SIMULATION ENVIRONMENT

NICHOLAS R. SIRERA
Summer 2012

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Mechanical Engineering and Nuclear Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Dr. Sean Brennan
Professor of Mechanical Engineering
Thesis Supervisor/Honors Adviser

Dr. Eric Marsh
Professor of Mechanical Engineering
Faculty Reader

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The driving simulator at Penn State University is currently in a state of disarray. Several attempts have been made in the past year to make bandage style fixes, each time raising more questions than they have answered. This thesis examines the prospects of designing an entirely new software architecture from scratch based on open-source freely available software. This decision was made not only to keep costs down, but to also take advantage of the large user bases and potential help available in the open-source community. The previous architecture, based on CarSim, was deemed too expensive and too prone to failure to pursue again. This research will examine an environment based on the Robotic Operating System (ROS) and the Gazebo Simulator. In addition to determining if this type of architecture is possible, this research will also investigate methods for converting LIDAR roadway scans into simulation models, and will attempt to make these models lifelike by applying physical characteristics. In addition, general conclusions as to the potential success of this new driving simulator architecture will be explored. Specifically, it was concluded that the method used in this paper to convert LIDAR point cloud data into a Gazebo simulation should not be recommended for further study. However, the underlying architecture of ROS and Gazebo simulator is strongly recommended for continued application in Penn State's driving simulator.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. Sean Brennan for his help as both my thesis advisor and my honors advisor for the past three years. In addition to providing a topic for my thesis, he insisted on finding one that I was both capable of tackling as well as interested in pursuing. I am very grateful for his insistence on pursuing a topic that stimulated my interest on a personal level, as it made the work significantly more enjoyable. In addition, he has always been willing to meet and discuss my project regardless of his incredibly busy schedule.

I would also like to thank Alex Brown and Jesse Pentzer for their help along the way. This thesis would have absolutely not been possible without Jesse's expertise in Ubuntu and Alex's expertise with all things related to vehicle simulations. They were always willing to step away from their own projects to help with mine and are the reason this paper was completed in time.

Chapter 1

Introduction

1.1 Background Information

Driving simulators have been present in mainstream society for several years and across several industries. The most prevalent use of these simulations can be seen in the enormous consumer market for video games, which can model anything from navigating a spaceship through an enemy attack, to skiing down the slopes of Aspen, to racing the streets of Los Angeles. As technology has progressed, the makers of these games have focused on creating more realistic and lifelike experiences. For example, the development of advanced physics engines like those encountered in the Unreal series created by Epic Games have revolutionized the approach taken by game designers. These gaming engines stressed the importance of lifelike simulation, from the lighting effects of cloudy skies to the bone-crunching whiplash of a quarterback being slammed into the turf. As is the case with most technologies, there was also a more practical and scientific benefit than simply the creation of a gaming universe. Researchers across several industries including automotive, aviation, and military/government have been studying and implementing various uses of driving simulation to enhance their understanding of real world situations without the associated cost and safety risks. Engineers at Volvo use the most advanced simulator in Sweden to study how drivers interact with their vehicles in order to design safety systems inspired by driver behavior

(Volvo, 2011). Student pilots practice risky maneuvers while veteran pilots practice approaches at unfamiliar airports using Microsoft's Flight Simulator (Beckman 2009). Even the FBI has licensed a version of the aforementioned Unreal Engine to develop a "multiplayer crime scene training simulation" to be used for training at the FBI Academy (Robertson 2012).

While simulation technology has progressed leaps and bounds from its inception, the field of combining remote robotic interaction with an offsite simulation environment is still in its relative infancy, dominated most prominently by the military's Unmanned Aerial Vehicle (UAV) effort. Attempts have been made in the open source community to develop standards for furthering research in these areas. Two such byproducts of these efforts are the Robotic Operating System (ROS) and Gazebo Simulator. This thesis will outline the first series of steps in using a ROS/Gazebo environment to develop a driving simulator that is capable of remotely operating a vehicle while simultaneously updating the simulation environment using information provided by sensors on the vehicle. This system could ultimately replace Penn State University's existing simulation package by providing an open sourced free solution that offers a higher level of performance.

1.2 Organization of this Research

Chapter 2 will contain a literature review of driving simulation and some examples of its history and usage in both regular society and in research. It will then explore the software applications used for this particular approach to robotics controlled driving simulations. Specifically, a background and introduction to the Robotic

Operating System (ROS) and its contributions to remote simulation control will be discussed. In addition, the Gazebo Simulator package and its compatibility with ROS will be explored. Finally, a summary of the current understanding of ROS/Gazebo systems will be presented along with specific goals for this particular application.

Chapter 3 will focus on the process involved with scanning a desired area or environment to be used as a simulation map and running the proper conversations to make it functional in a ROS/Gazebo framework. It will also provide specific details on which software packages were utilized to fulfill these requirements. Chapter 4 will discuss the process of taking the results from Chapter 3 and implementing them into a ROS/Gazebo world.

Finally, Chapter 5 will present a summary of this particular approach to using ROS/Gazebo in creating a simulation world and will discuss any setbacks, limitations, or conclusions drawn from the research. It will also include some topics for further study that were spawned during the course of this research.

Chapter 2

Literature Review

2.1 Overview

This literature review will begin by examining driving simulations and their prevalence in society. It will discuss a history of the simulation experience, dating back to early video games and tools that were only suitable for recreational experiences. It will then transition into the use of driving simulations as a research and educational tool, specifically how evolving technology allows for more realistic simulation experiences. The first section will conclude by examining the current research applications for driving simulators.

The next two sections of the literature review will focus on the software packages used in this research: Robotic Operating System (ROS) and Gazebo Simulator. A history of each software package along with a basic explanation of the operating principles behind each package will be discussed.

The next section will discuss the combination of these two packages into what is known as a ROS/Gazebo environment and give a few examples of current projects using this architecture.

The final section of this literature review will include a brief summary of the key points that pertain to this research project, and a list of goals based upon the furthering of the field's current work with ROS and Gazebo will be presented.

2.2 Driving Simulation

To examine the history of visually based simulation, it is not necessary to focus exclusively on the high-level engineering and research markets. In fact, the majority of people have experienced this field of technology from the comfort of their own homes, not in an expensive laboratory underground. Almost as quickly as graphical user interface (GUI) based computers were hitting the market, companies like Atari were producing the world's first video game systems. As technology progressed over the next few decades, a radical shift in the science behind these games can be clearly documented. While some games, like Nintendo's Mario Kart, focused very lightly on reality at the expense of a more fun user experience, some more recent games like Microsoft Xbox's Forza Motorsport have taken a sophisticated approach to modeling real life vehicle dynamics in their games (Togelius et al., 2007). EA Sports successful NHL franchise has made stunning simulation improvements over the last few releases, including a revamped physics collision engine in 2012 which promised that every single body check would evoke a different and realistic reaction from the players involved. In the 2013 edition, the game tackles another area of simulation physics with a new momentum based skating engine that flawlessly mimics the way a player interacts with the ice using steel blades just a few millimeters wide (Sarkar 2012). While these games are mainly for sport, the first realistic simulation game designed for educational as well as recreational use was actually developed 35 years ago in 1977 and holds the title as the world's longest-running PC game series of all time (Remo, 2009). Microsoft's popular Flight Simulator, officially released in 1980, invented a whole new industry of training for both military and civilian alike using a simulated lifelike experience, while subjecting the user to none of the risks associated with on-the-job training. As Beckman (2009) states, the original version of the game was run on computers lacking the processing capabilities required to portray a very lifelike scenario and was therefore viewed by pilots as nothing more than a game. However, since the turn of the century, both the computer hardware and gaming software have advanced enough to simulate a fairly realistic flight experience. She goes on to claim that over 85% of pilots have indicated that they use Flight Simulator to practice approaches for new airports, and 88% of those pilots have found the software effective in this effort. It is clear that these simulations are useful in predicting real life behavior and should continue to be studied in depth.

Several hundred studies have been performed using driving simulators to examine various areas of automotive behavior. Many of these focus on assessing human behavior, using the simulator as a tool to allow for safer and easier data acquisition. For example, researchers at the Institute for Psychosocial Medicine in Sweden published a study in 2004 examining the effects on performance and alertness suffered by drivers commuting home after a night shift. They used a driving simulator modeled after the Volvo 850 that was capable of simulating acceleration in three dimensions using roll, pitch and liner lateral motion. The study was conducted using participants who regularly worked a night shift and who would report to the simulator immediately after the completion of a shift. The participants in their study were involved in 18 simulated accidents when driving after a night shift versus just two when driving after a full night’s sleep. In a study such as this, the benefits of using a simulator versus a real car are obvious, as it would be incredibly careless and dangerous to test sleepless individuals in a real life vehicle. A graph of the results of this study can be found in Appendix A (Åkerstedt et al., 2004).

In addition to studies focusing on human behavior, there have been many others that focus on the data collection and terrain modeling of simulated roadways. Researchers at Penn State's Intelligent Vehicles and Systems Group published a study in 2011 that examined this process of transforming raw collection data into a suitable simulation model (Varunjikar et al., 2011). As the study points out, most research in the field focuses on idealized roads, where as this particular study focused on optimizing the comparison of simulation-predicted results to actual measurements from the road. Using data collected from LIDAR scans of roadways, Varunjikar attempted to find the best way to filter the raw point cloud data into a suitable 3D road representation. The study compared pitch and roll data taken from IMU measurements to those produced by a simulation to determine the accuracy of its filtering process. Varunjikar's terrain modeling approach was so accurate that simulated emergency maneuvers like lane changes and sudden breaking events could actually be distinguished from normal driving behavior. This type of terrain modeling represents a significant improvement to driving simulators based upon idealized roadways. A graphical depiction of this emergency maneuver detection can be seen in Appendix B.

Another field of interest involves utilizing driving simulators to help model human behavior for the programming of autonomous vehicles. In 2001, researchers from Northeastern University published a study in which the goal was to mimic human behavior in a two-lane rural environment (Al-Shihabi and Mourant, 2001). On a fundamental level, the group divided human driving into four units of focus. The

Perception Unit served to define how the vehicle would perceive its environment in both global and local terms. The Emotions Unit defined how the model would respond emotionally to its changing environment. The Decision-making Unit (DMU) would take input from the Emotions Unit and attempt to find suitable courses of action based upon the environment. Finally, the Decision-implementation Unit (DIU) attempted to implement decisions when a traffic condition arose. This framework can be interpreted pictorially in Figure 2-1.

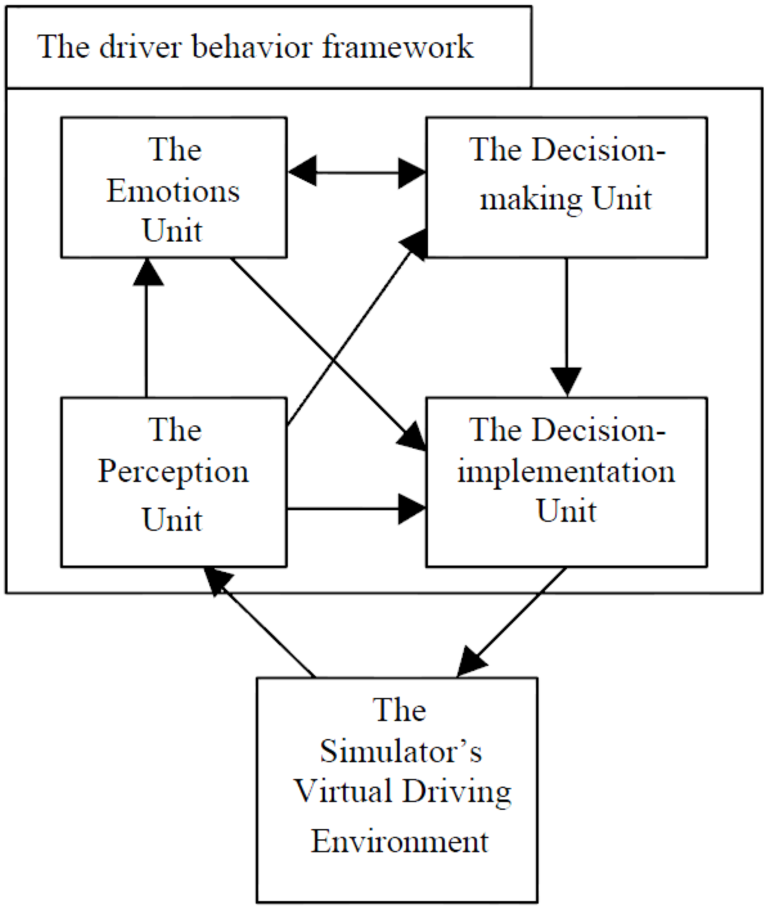

The driver behavior framework
The Emotions Unit
The Decision-making Unit
The Perception Unit
The Decision-implementation Unit
The Simulator's Virtual Driving Environment

Figure 2-1: Driver behavior framework (adopted from Al-Shihabi and Mourant, 2001)

When applied to an autonomous vehicle in a simulation environment, the relationship between the driver behavior model and the rest of the environment can be shown in Figure 2-2.

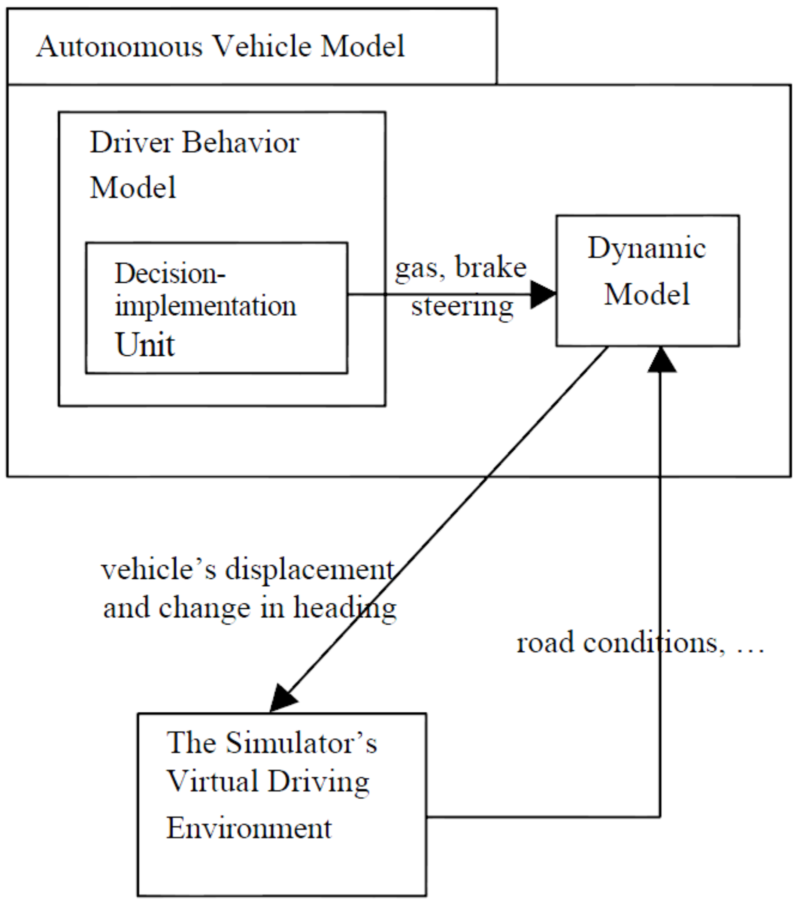

Autonomous Vehicle Model
Driver Behavior Model
Decision-implementation Unit
gas, brake steering
Dynamic Model
vehicle's displacement and change in heading
road conditions, …
The Simulator's Virtual Driving Environment

Figure 2-2: The Decision-implementation Unit within the total simulation model (adopted from Al-Shihabi and Mourant, 2001)

This research resulted in more human-like autonomous vehicles and a virtual simulation environment that was more realistic and less predictable. In turn, it helped to lay a foundation for future work modeling human driving behavior for replication in autonomous vehicles.

2.3 Robotic Operating System (ROS)

As the field of robotics has evolved over the last few decades, one of the most fundamental concerns has been establishing a set of standards for writing robotics software. As the scale and scope of robotics continues to grow, the ever varying hardware platforms make it difficult to reuse code from one project to another. Also, the sheer size of the code alone can be intimidating, as each particular robot requires programming from driver level software on through perception, abstract reasoning and beyond (Quigley et al. 2009). A wide variety of software frameworks have been created to meet these needs, however no one package has successfully accomplished the goal of becoming an all encompassing solution. A few of these solutions include Microsoft's Robotic Developer Studio (RDS), Gostai's Urbi, Evolution Robotics' ERSP, National Instruments' LabVIEW, and Willow Garage's Robotic Operating System (ROS) (Guizzo 2010).

The Robotic Operating System (ROS) was originally written under the code name "switchyard" by Morgan Quigley as part of the Stanford AI Robot (STAIR) project in 2007. In 2008, development shifted primarily to the Personal Robots Program at Willow Garage, where it is still worked on and distributed today (Willow Garage, 2012). While it was not intended to be the best framework for all possible robotic applications, it was designed to meet the following philosophical goals (Quigley et al. 2009):

1. Peer-to-Peer
2. Tools-based
3. Multi-lingual

4. Thin
5. Free and Open-Source

In brief summary, ROS was chosen to utilize peer-to-peer connections instead of a central server design to limit unnecessary traffic flow across slower wireless links, which are eliminated with its service/master architecture. The developers chose a tools-based microkernel design as opposed to a monolithic development and runtime environment with the purpose of improving stability and complexity management. Multi-lingual support was included to provide flexibility and freedom of choice to robot designers who use a variety of programming languages. ROS is intended to be language neutral, supporting C++, Python, Octave, LISP, and many others. The developers of ROS also designed it to be "thin," which means that virtually all complexity is located in standalone libraries and not in the ROS code itself. By separating the dependencies of codes such as driver and algorithm development from that of the ROS code, ROS allows for easier code extraction and reuse beyond the original intent of the programming. Finally, the creators of ROS believed that the best way to facilitate debugging at all levels of the software stack was to create and distribute ROS as a free and open-sourced operating system. In addition to better debugging, open-source software promotes quick adoption by a much broader usage base, which in turn helps it achieve an industry standard status.

On a functional level, the ROS operating system can be broken down into four fundamental concepts (Quigley et al. 2009):

1. Nodes
2. Messages

3. Topics
4. Services

In agreement with its modular based design, the fundamental processes that perform computations in a ROS system are called “nodes.” A system typically consists of several nodes that communicate with each other in accordance with the peer-to-peer design philosophy. Specifically, nodes communicate with each other by sending “messages.” The message itself is a data structure that has a very strict format and can include integers, floating points, Booleans and constants among others. The way a node actually sends a message is by publishing it to a pre-established “topic.” Nodes can publish and subscribe to multiple topics, which are just strings. In addition, topics can have multiple different nodes publishing and subscribing to them. This creates a system in which the publishers and subscribers are generally not aware that each other exist, but rather are linked together by these shared topics. Because topics are not useful for synchronous transactions, “services” are used instead. A service is a string name along with two rigorously defined messages (one for request and one for response). Unlike topics, any particular service can only be advertised by one specific node. These four concepts make up the essence of communication using ROS, and allow for it to satisfy several of the five design philosophies.

2.4 Gazebo Simulator

As robotic applications have increased in both number and complexity, it has become obvious that not every change, feature, or strategy can be field tested in a reasonable amount of time. Simulators have solved that problem by developing virtual worlds that closely model the physics of real life, while simultaneously allowing for testing and data collection in a reliable way. A few of the more useful 3D simulation software packages include Festo's COSIMIR, Cyberbotics' Webots, Mechanical Simulation Corporation's CarSim, and Willow Garage's Gazebo.

The Gazebo Simulator project started in the fall of 2002 at the University of Southern California. It was written during the time of the Player/Stage project by Dr. Andrew Howard and his student Nate Koenig, and was designed to fulfill the need for a 3D simulator capable of simulating a population of robots, sensors, and objects, while also generating realistic sensor feedback and physically lifelike interactions between objects (Gazebo, 2012). According to its co-founder Nate Koenig, what separated Gazebo from its predecessor, the Stage project, was the need for a simulator capable of handling robotic vehicles in outdoor applications. It is able to reproduce the mass, velocity, friction, and other object attributes in order to realistically simulate those objects being pushed, pulled, or knocked over. In addition, the robots used in the simulator are dynamic structures utilizing rigid bodies connected by joints. They can accurately interact with the landscape in which they are placed, including interaction with any and all user defined objects. Finally, all aspects of the landscape environment can be controlled as well, including lighting conditions as well as friction coefficients (Koenig and Howard, 2004). Although the Gazebo simulator offers these sophisticated features, it is still limited in various areas, including the physics modeling of soil, sand, and grass, and in its ability to incorporate fluid dynamics and thermodynamics into its models.

Despite these limitations, Gazebo is still an excellent choice for outdoor robotic simulation.

On an operational level, the Gazebo simulator was designed around the philosophy that new robots, actuators, sensors, and arbitrary objects should be easy to create and integrate with the software. To achieve this, a very simple Application Programming Interface (API) is maintained along with open source libraries which handle the visualization and physics simulation. A graphical interpretation of this architecture can be seen in Figure 2-3.

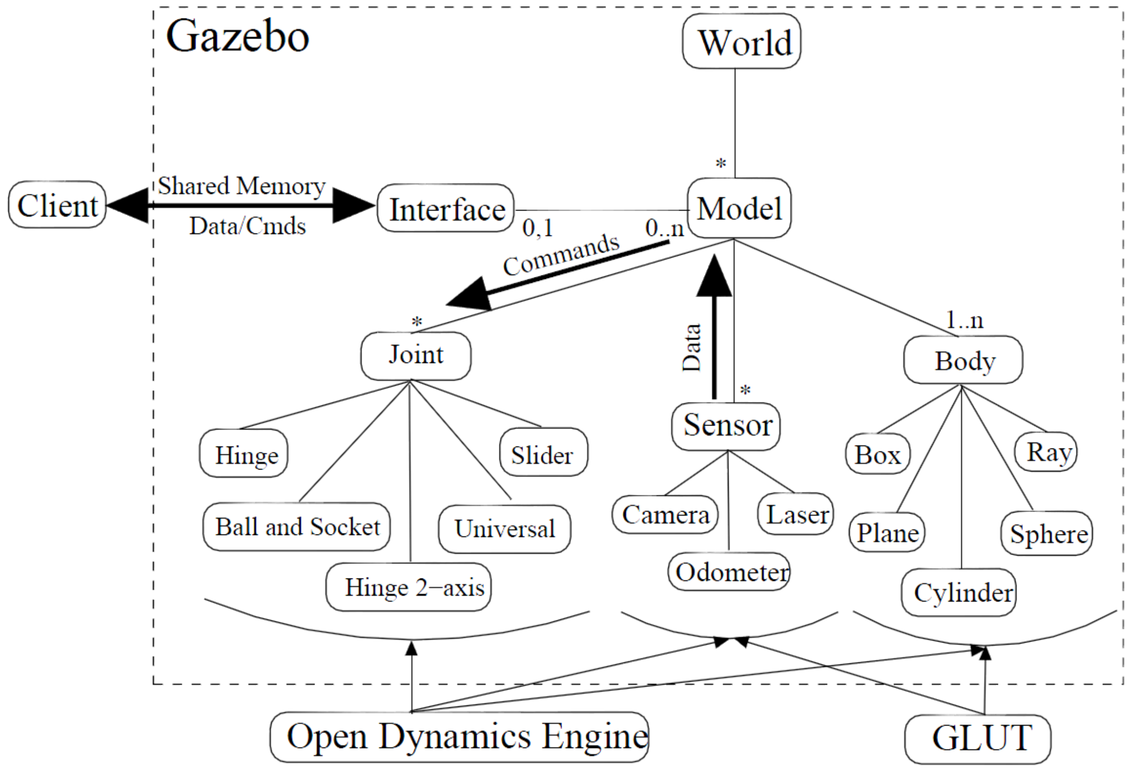


Figure 2-3: General Structure of Gazebo components (adopted from Koenig and Howard, 2004)

As with many popular simulation packages, Gazebo utilizes the Open Dynamics Engine physics engine created by Russell Smith. The ODE is a free library for simulating articulated rigid body dynamics on a level of industrial quality. According to its creator, the ODE is good for simulating articulated rigid body structures, was designed to be used in interactive or real-time simulation, uses a highly stable integrator (to avoid exponential growth of simulation errors), has hard contacts (colliding bodies do not penetrate one another), and has a built-in collision system (Smith, 2006). Gazebo expands upon ODE by allowing for the creation of both normal (using ODE) and abstract (using Gazebo) objects. By separating the abstraction, it is possible to interchange the physics engine should another be more suitable (Koenig and Howard, 2004).

While the primary function of Gazebo is to simulate dynamics, which can occur with or without a graphical user interface (GUI), its success has been largely attributed to its elegant interface. The visualization in Gazebo is controlled by OpenGL and OpenGL Utility Toolkit (GLUT). OpenGL is a platform independent standard library for creating 2D and 3D interactive applications, while GLUT is an independent toolkit for OpenGL applications that utilizes a window system. In essence, OpenGL renders the visualizations, which are displayed to the user via GLUT. GLUT is also platform independent, and can interact with the most common input methods such as keyboards and mice. This visualization system is especially useful because many of its key features are already implemented in graphics hardware, allowing the CPU to spend more of its resources on the computationally demanding dynamics engine (Koenig and Howard, 2004).

The collection of models and sensors make up “The World” in a Gazebo environment. The models are directly connected to the dynamics engine, and are divided into stationary (buildings, roads, etc.) and dynamic (robots, cars, etc.). A model can be defined as any object that has a physical representation, and can be as simple as one rigid body. The models are made up of bodies, joints, and interfaces. Bodies are the fundamental building blocks of models and assume all the physical properties of an object such as mass, friction, and color. They take the form of lines, planes, spheres, boxes, and cylinders. Joints are the instruments by which bodies are connected together. As in real life, many different joints exist to form the kinematic and dynamic relationships between bodies, such as hinge joints, slider joints, and universal joints. Finally, interfaces allow for the client programs to access and control models. They can instruct a joint to move via the application of a force, or they can instruct a sensor to collect and report back data. Sensors are kept separate from the dynamics engine, as they only collect and emit data. In Gazebo, they are abstract components until applied to a physical model. Sensors can be used for a number of useful purposes, such as measuring distance traveled, measuring distance between the sensor and other objects, and viewing the world through the eyes of the model, all of which would be very useful in a remote control simulation environment.

2.5 ROS/Gazebo Practical Application

Both ROS and Gazebo have been implemented in a number of successful research and industry projects, including several in which they are combined to form a ROS/Gazebo environment. In the case of industry, ROS has been making headway having been utilized as the middleware in a number of commercial products including Aldebaran’s NAO humanoid and Meka Robotics’ systems. Most surprisingly, however, was the Southwest Research Institute's (a privately owned R&D organization located in San Antonia, Texas) agreement with Motoman (one of the biggest industrial robot makers in the world) to develop a ROS interface for Motoman's SIA20 7-axis robot (Bouchard, 2011). Prior to this announcement, private robotics companies would create their own proprietary OS and controller. Motoman is betting that the open-source nature of the ROS community could prove to be a valuable asset to its customers. In the research field, Gazebo has also witnessed a flurry of activity. In the traditional robotic navigation sense, several groups have undertaken projects utilizing Gazebo as their primary simulation and dynamics engine. For example, a team at the University of Freiburg has been studying ways to make mobile robots self aware of system faults and to identify where these faults occur (Plagemann, 2006). They rely heavily on Gazebo to test their detection algorithm. As shown in Figure 2-4, the image on the left simulates a robot that has collided with an undetected object, while the image on the right shows a robot that has collided with a glass door undetected by its sensors.

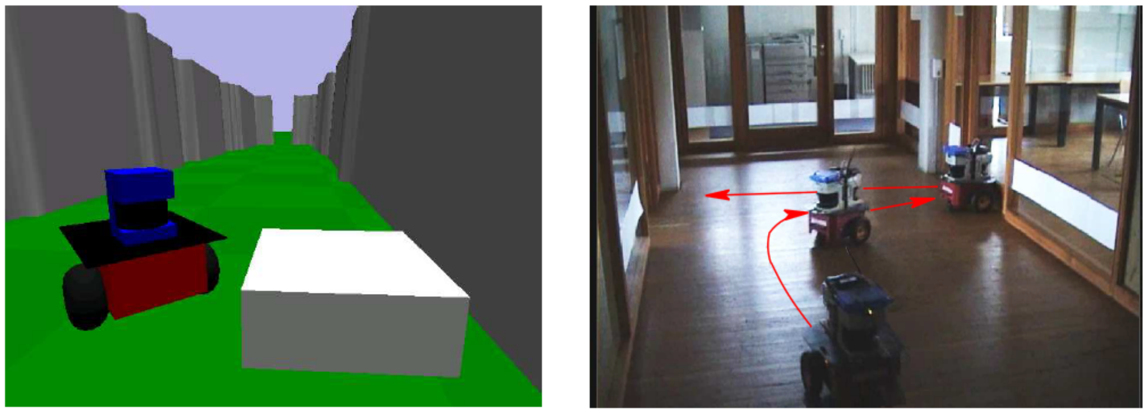


Figure 2-4: University of Freiburg's detection algorithm in action (adopted from Plagemann, 2006)

In addition to this traditional research, some project teams have taken a more creative approach to utilizing Gazebo. The Intelligent Autonomous Systems group at the Technische Universität Műnchen has been working on a project that is testing the limits of what is possible in Gazebo. Their goal is to extend Gazebo's capabilities beyond those of simple navigation and into such realms as RFID scanning, enhanced robotic manipulation (recognizing, measuring, and determining the best grip position to grasp objects), and enabling cognitive processing directly in the Gazebo simulation middleware (Rusu et al., 2007). Figure 2-5 gives a hypothetical depiction of what one of these futuristic robots could look like, using new sensors to detect a range of items such as the RFID compatible cupboard and the half full glasses on the kitchen top.

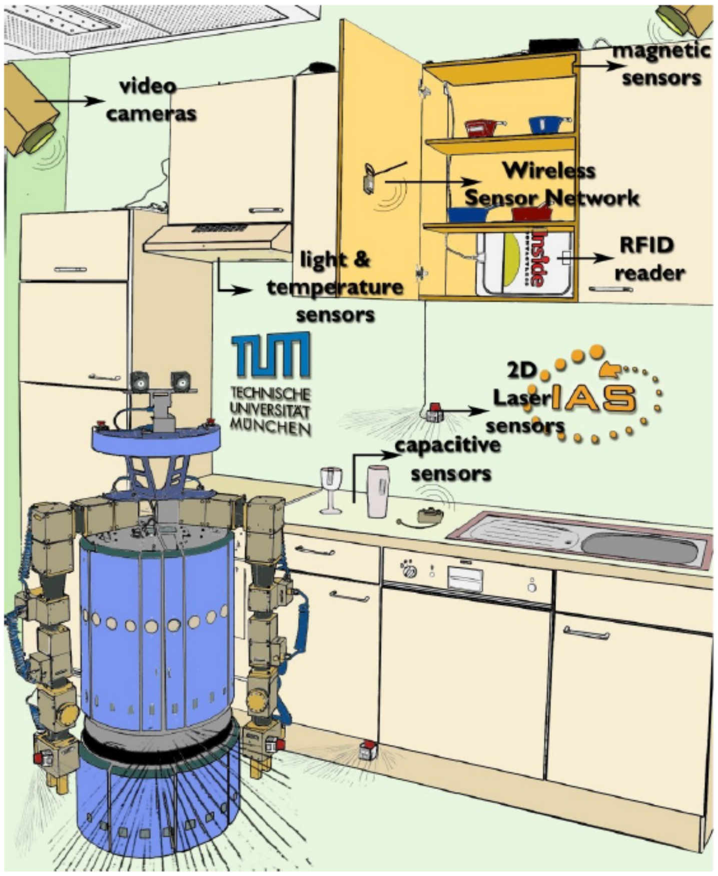


Figure 2-5: A possible design for the AwareKitchen (adopted from Rusu et al., 2007)

While these projects represent great strides forward for both ROS and Gazebo, the projects of most interest to this thesis are those that combine the two software packages into what is known as a ROS/Gazebo environment. These environments utilize a specially designed version of Gazebo that is intended to work seamlessly with ROS. This environment has been the platform of choice for many recent projects, including the research performed in this thesis. At Georgia Tech, robotics students have been working on Autonomous Surface Vessels (ASVs) to submit in the AUVSI RoboBoat Competition for each of the past three years (Pickem et al., 2011). As the name implies, the AUVSI RoboBoat Competition is an annual robotics competition held each July by the

Association for Unmanned Vehicle Systems International. In July of 2012, they held their 5th annual event, which presents students with an aquatic obstacle course through which the students are to race their self-designed ASVs (AUVSI, 2012). For the 2012 competition, the team from Georgia Tech switched the entire software architecture of their ASV to a ROS/Gazebo environment. Making such a switch achieved quite a few benefits; in the words of the Georgia Tech team:

> The biggest advantage of ROS though is the straightforward switch between simulation and real hardware. All that has to be changed are the drivers supplying the sensory inputs, i.e. whether the sensor data is published by Gazebo or by real hardware sensors. ROS itself and all modules that process sensor data are agnostic to the underlying low-level driver architecture meaning that once sensor data is published in the form of ROS messages, all higher software layers can use said data and are not concerned with whether the data was created by a simulation or by physical sensors. The hardware abstraction layers in the form of drivers is therefore completely transparent to all higher level layers. The only purpose of hardware drivers is to read data from sensors and publish that data as ROS message (Pickem et al., 2011).

As they pointed out, the switch was made primarily for the benefits of switching between real world and simulation seamlessly, which is an essential part of this thesis project. To demonstrate just how radically the ROS/Gazebo environment changed their robot, consider the comparison between the electronics box from the 2011 robot (shown left) to the 2012 robot (shown right) depicted in Figure 2-6.

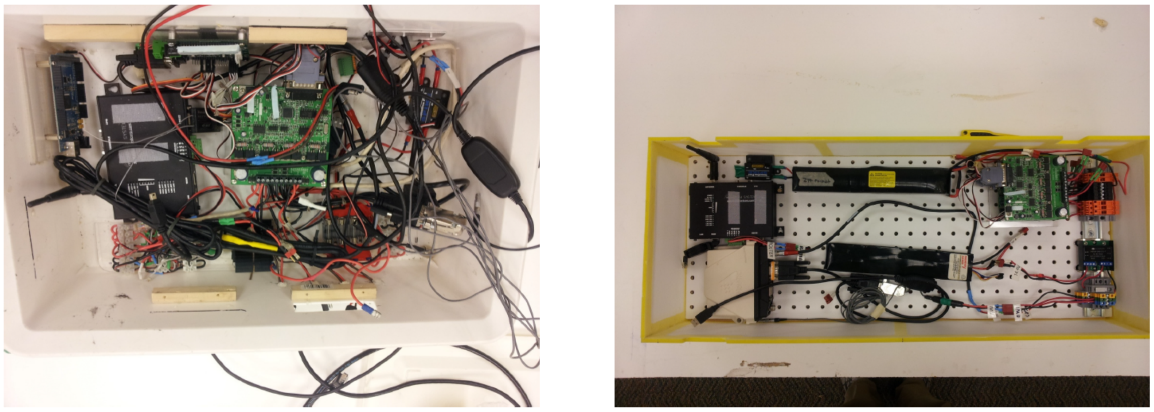


Figure 2-6: The 2011 (left) and 2012 (right) electronics box for Georgia Tech's ASV (adopted from Pickem et al., 2011)

The ROS/Gazebo software architecture significantly simplifies the hardware required to perform the same functions of processing simulation and real world sensor data. In addition to improvements in hardware, the team noticed strong improvements in their simulation environment. Because ROS acts as an interface to route data and Gazebo is capable of simulating sensor data like GPS, IMU, and camera images in addition to physical properties like friction and mass, the Georgia Tech group was able to test their code for position and velocity estimation and control, obstacle avoidance, global waypoints, blob tracking, template matching and point cloud generation. They were able to achieve all of this before deploying any of these systems on the actual boat, due in great part to the ROS/Gazebo setup.

2.6 Summary and Research Goals

This literature review has provided an introduction to several aspects of importance to the rest of this paper. First, an introduction into simulation, both generic and with respect to automotive vehicles has been provided along with current areas of research and a general reasoning of why this field is an important one to investigate further. Second, an introduction into the two software packages ROS (Robotic Operating System) and Gazebo Simulator was given which included a history of their development and an overview of their general methodology. In addition, several projects utilizing both of these software packages were examined, culminating in an in-depth look at the research potential of combining the two into a ROS/Gazebo environment.

This research aims to solve the first series of obstacles in constructing a ROS/Gazebo environment to be used as part of a solution to fix Penn State University's ailing driving simulator. Specifically, this paper will focus on the preliminary steps of converting point cloud data obtained from a LIDAR scan into a simulation map to be used in Gazebo. This research focuses on processing data that was previously collected (not processing live data), but the insight gathered from this research should provide a solid foundation for further work on this driving simulator project.

Chapter 3

Conversion of Point Cloud Scan to Collada Mesh

3.1 Obtaining Point Cloud via LIDAR Scan

While the physical act of obtaining a Light Detection and Ranging (LIDAR) scan is not covered within the scope of this paper, it is important to briefly discuss the purpose of such a step. Considering once again the long-term goal of this research, which is to construct a real-time simulation and remote control system, it is important to use sensors that can produce accurate data in a variety of environments. It is also important to use sensors which produce real-time data that requires as little preprocessing as possible. LIDAR accomplishes all three of these goals. First, LIDAR scanning produces range data that can be used to construct terrain models within 1-2 cm of accuracy. Second, LIDAR scanning does not depend on ambient lighting and therefore produces this accurate data regardless of lighting conditions. Third, the LIDAR sensor produces geometric data in the form of a 3D point cloud, which is essential for real-time processing of the scan (Rekleitis et al., 2009).

The data used in this thesis was produced from a LIDAR scan of Cheat Lake, West Virginia and covers a stretch of road a few hundred meters long. It was given in Stanford (*.ply) format, and is essentially a three columned list of geometric points formatted in ASCII. While the actual file contains 2,031 elements, an abbreviated list of the first 20 elements can be seen in Table 3-1.

Table 3-1: Abbreviated Cheat Lake 3D Point Cloud Data

ply format ascii 1.0 element vertex 2031 property float x property float y property float z end_header		
0	0	0
-6.55829	4.429809	0.415514
-12.9737	9.047907	0.852932
-19.4399	13.68936	1.310421
-25.7991	18.39562	1.770145
-32.0649	23.15353	2.254174
-38.3692	28.22305	2.60648
-44.5458	33.18459	3.081154
-50.6534	38.19985	3.529472
-56.5521	43.15836	3.944815
-62.7542	48.62357	4.387047
-68.6404	53.85921	4.870702
-74.4937	59.22307	5.282593
-80.2803	64.77343	5.682815
-85.9339	70.25742	6.130975
-91.5566	75.8537	6.624161
-97.0901	81.50518	7.056619
-102.504	87.28787	7.462627
-107.892	93.15742	7.914748
-113.111	99.03428	8.376996

All examples and screen shots provided in Chapters 3 and 4 were produced using this data, and they were produced under the assumption that they would contain enough detail to handle future simulations.

3.2 File Processing in Blender

An intermediary software package was used to convert the point cloud data given in the Stanford (*.ply) file to a Collada Mesh (*.dae) file recognized by Gazebo. While there are several potential candidates, ultimately Blender was chosen for this research. Blender is a 3D computer graphics software package useful for visual effects, interactive 3D applications, and animated films. Most importantly, however, it is open-source and easily attainable as a free download. Following the same reasoning behind the selections of ROS and Gazebo, it was important to select an open-source file processor, both for reasons of cost and of maximum compatibility with future updates to this simulation system. Blender accomplishes both of these, while providing sufficient documentation and tutorials generated by its large user base.

The first step in converting a Stanford point cloud to a Collada Mesh is to skin the point cloud, thereby creating a surface. While Blender version 2.62 does not offer native support for such a skinning, there is a free Python plug-in script called “Point Cloud Skinner (v.0.16)” created by internet user “Hans.P.G.” available on the BlenderArtists.org website (Hans.P.G., 2012). Installation of the plug-in is very straightforward. Simply navigate to the “File, User Preferences” menu, then select Add-ons, and Install New Add-on. Locate the downloaded Python script, and install. Select the checkbox next to the Point Cloud Skinning script to make sure it is enabled. Now that Blender is capable of skinning a point cloud, the next step is to import the Stanford Point Cloud. A visual tutorial is shown in Figure 3-1.

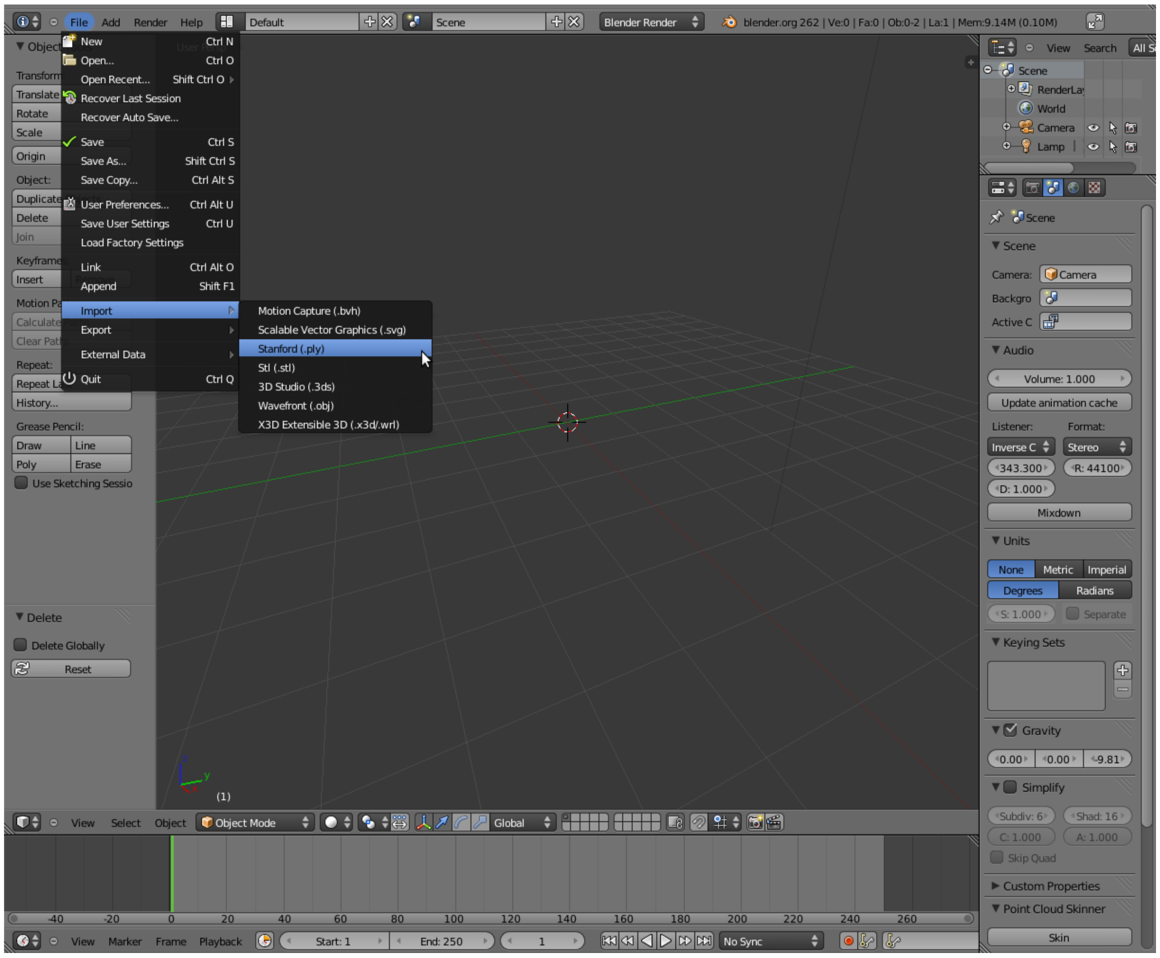


Figure 3-1: Importing a point cloud into Blender

After selecting the file, the point cloud should appear in Blender's main window. For the Cheat Lake example, see Figure 3-2.

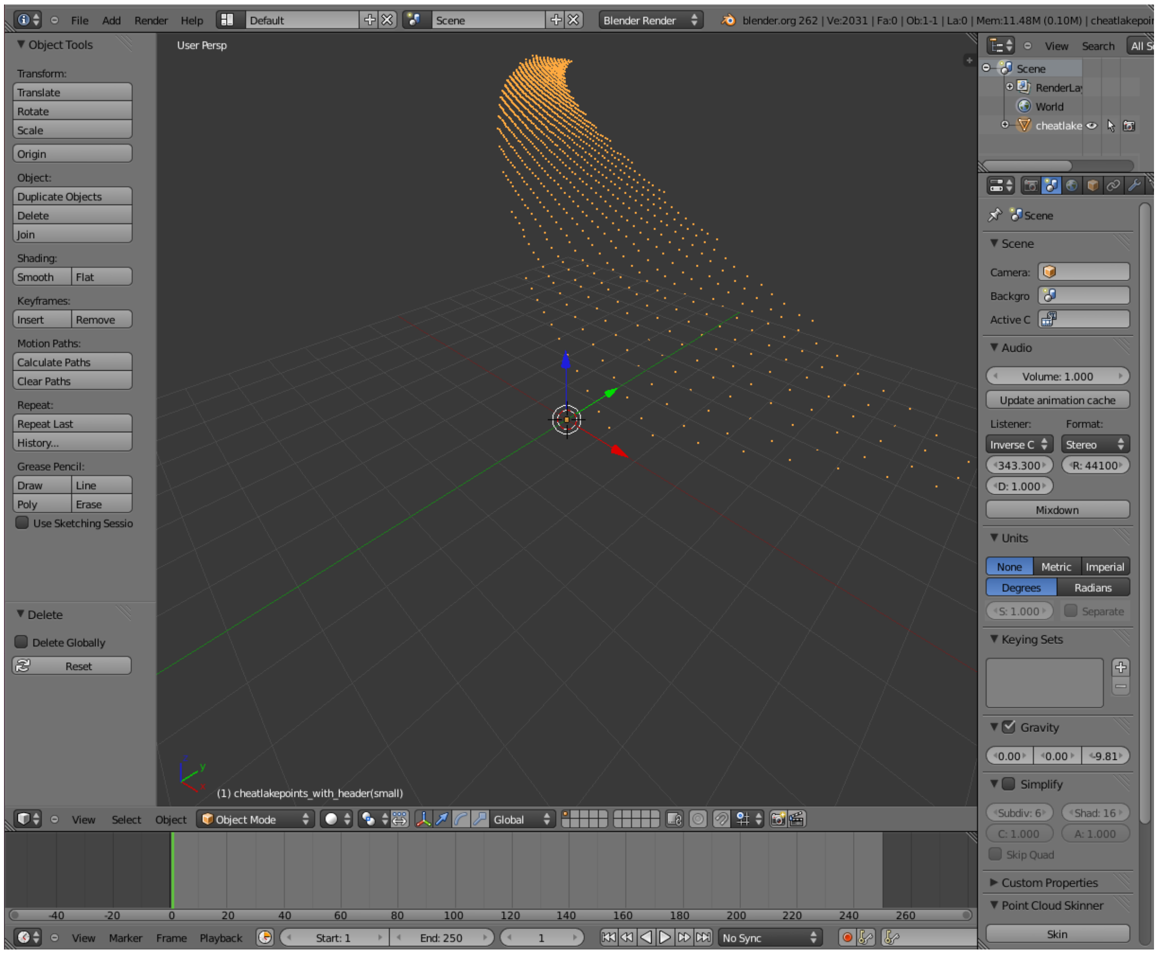
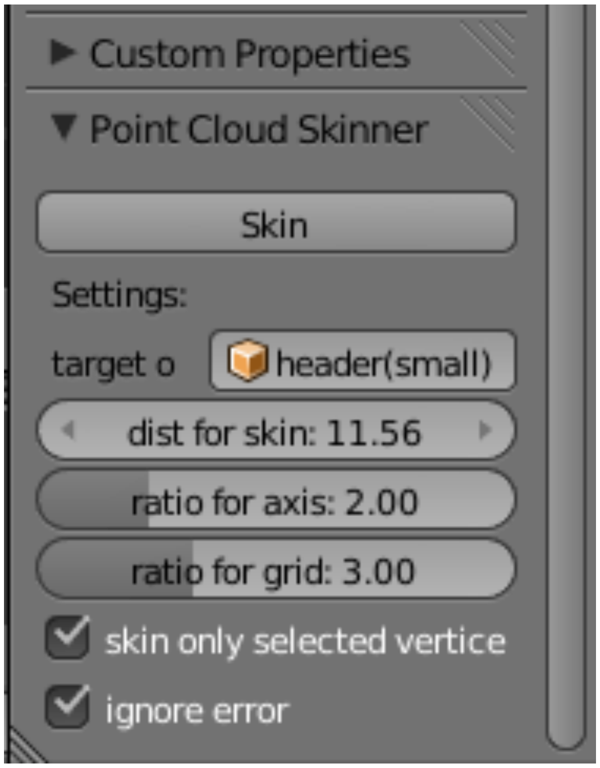


Figure 3-2: Cheat Lake point cloud map

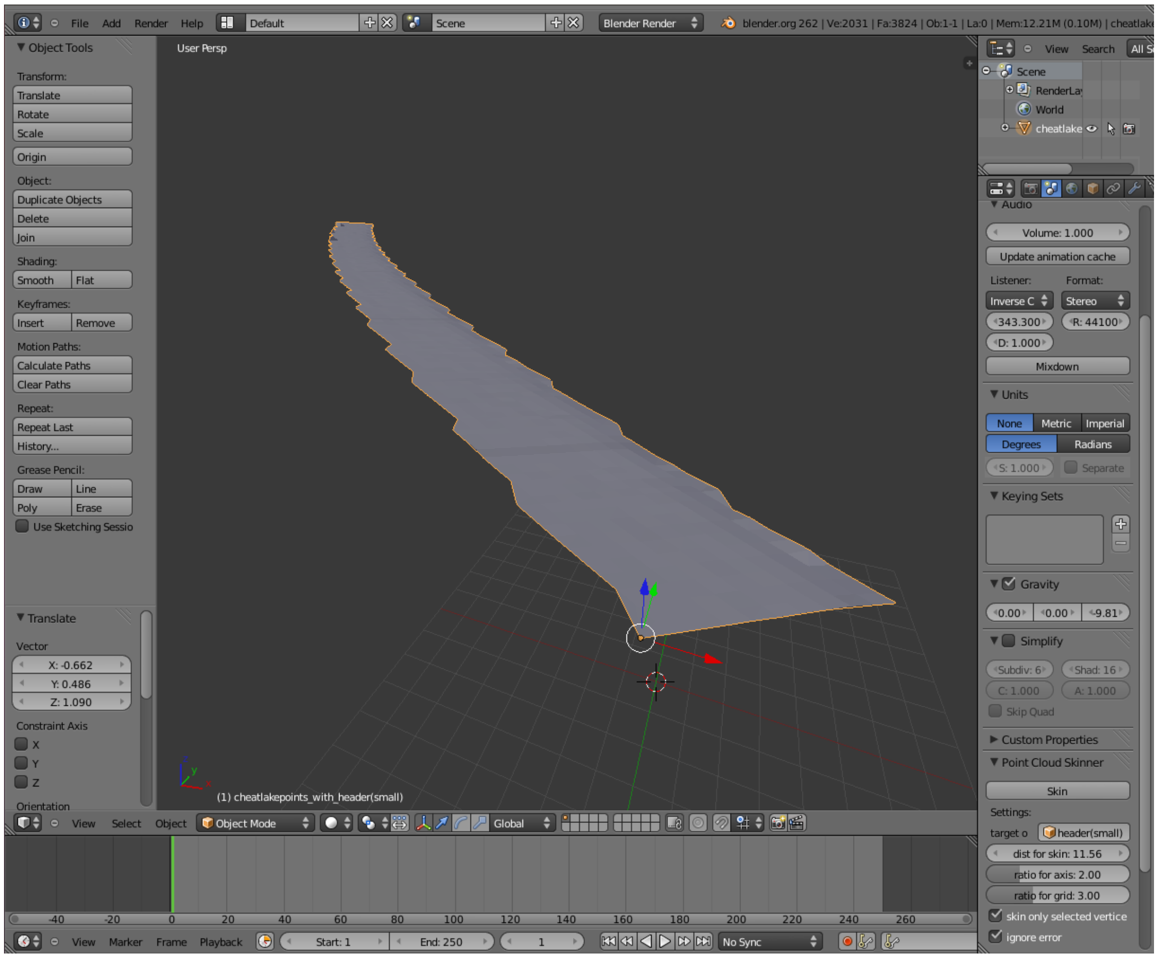
Now that the road's point cloud scan has been imported into Blender, the next step is to perform a point cloud skin that will turn this collection of 3D points into a surface. The recently added Point Cloud Skinner menu is visible in the bottom right hand corner of Figure 3-2. After expanding the menu, several options for the skinner are available. A close-up view can be seen in Figure 3-3.



Settings:
target o: header(small)
dist for skin: 11.56
ratio for axis: 2.00
ratio for grid: 3.00
✓ skin only selected vertice
✓ ignore error

Figure 3-3: Point Cloud Skinner script settings

It is essential to select the proper point cloud from the "target" menu. After this has been selected, the only other setting of importance is the "distance for skin." This setting controls the radius the skinner uses to connect points together. A greater distance will result in a better skin; however it will significantly increase the processing time necessary to generate the skin. A distance around 10 worked well for the Cheat Lake point cloud. The results of this point cloud skin can be seen in Figure 3-4.



blender.org 262 | Ve:2031 | Fa:3824 | Ob:1-1 | La:0 | Mem:12.21M (0.10M) | cheatlak

(1) cheatlakepoints_with_header(small)

Translate
Vector
X: -0.662
Y: 0.486
Z: 1.090
Constraint Axis
X
Y
Z
Orientation

Volume: 1.000
Listener: Inverse C
Format: Stereo
343.300
R: 44100
D: 1.000

Units: None [✓] Metric Imperial
Degrees [✓] Radians
S: 1.000
Separate

Gravity ✓
0.00 0.00 -9.81

Simplify
Subdiv: 6
Shad: 16
C: 1.000
A: 1.000
Skip Quad

Settings:
target o header(small)
dist for skin: 11.56
ratio for axis: 2.00
ratio for grid: 3.00
✓ skin only selected vertice
✓ ignore error

Start: 1
End: 250
1

Figure 3-4: The Cheat Lake road point cloud after skinning

Because of compatibility issues between the Point Cloud Skinner, the Collada Mesh Export tool, and Blender versions 2.62 and 2.63a, it is necessary at this point in the process to save the skinned point cloud as a blender (*.blend) file. In this example the file was saved as “**cheatlaketurn.blend**”. These compatibility issues will be explored further in the next section.

3.3 Preparing the Collada Mesh

The preparation of the Collada Mesh to be used in Gazebo is relatively straightforward compared to the previous steps. While previous versions of Blender supported Collada Mesh Export via the use of plug-ins, version 2.62 does not support Collada Meshing at all. However, the very recently released version 2.63a (Blender, May 2012) provides native support for Collada Mesh Exportation. While version 2.63a supports Collada Mesh Export, it does not support the Point Cloud Skinner plug-in. It is for this reason that dividing the conversion into two separate steps, handled by two different versions of Blender is necessary. Using Blender 2.63a, the skinned point cloud file created in the previous steps (**cheatlaketurn.blend**) is opened as an existing project. To convert the blend file into a Collada Mesh (*.dae), navigate to the export window and select "COLLADA (*.dae). Name the file and select a desired save destination (**cheatlaketurn.dae**). This process can be seen in Figure 3-5.

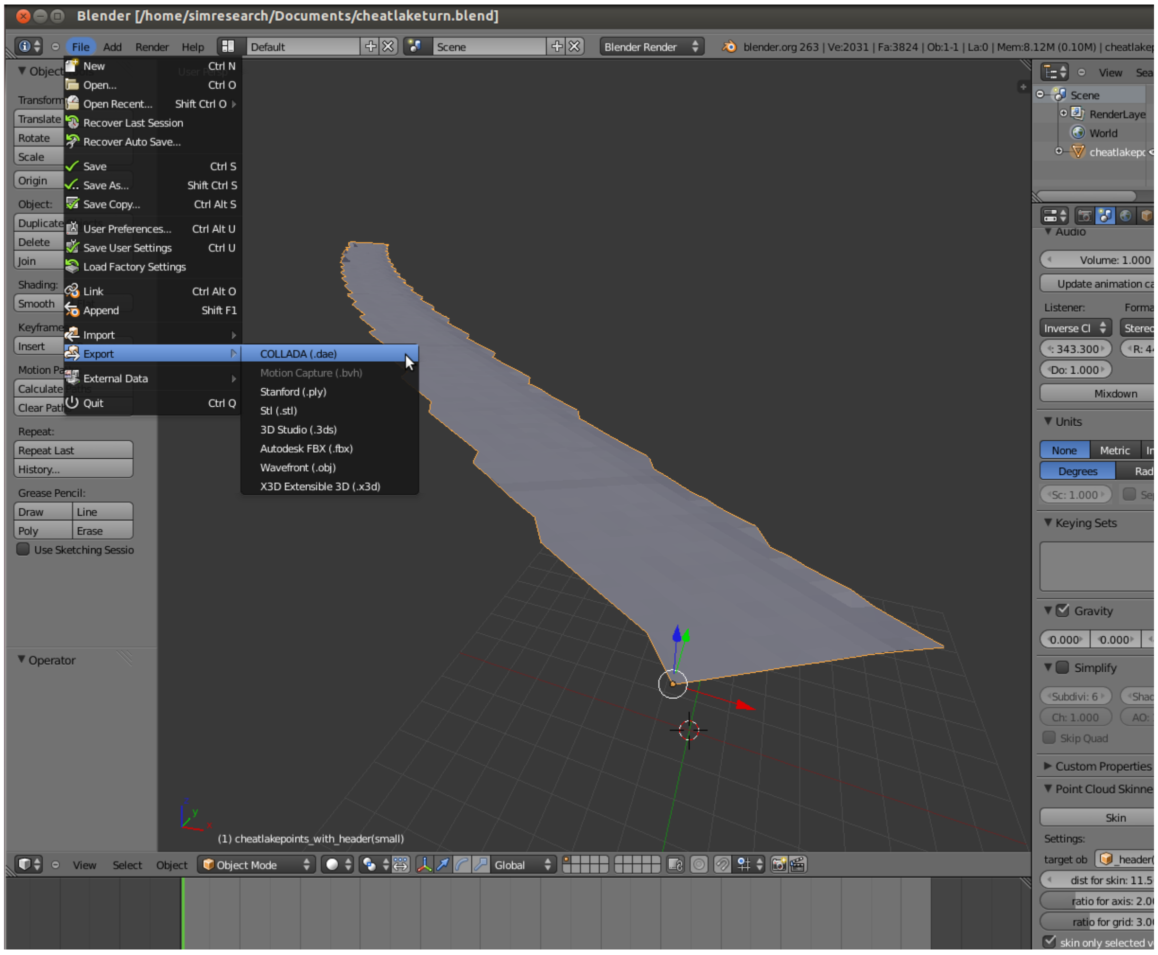


Figure 3-5: Exporting road to Collada Mesh

At this point in the process, the work using Blender has been completed. The road scan is now ready to be used as input into Gazebo.

Chapter 4

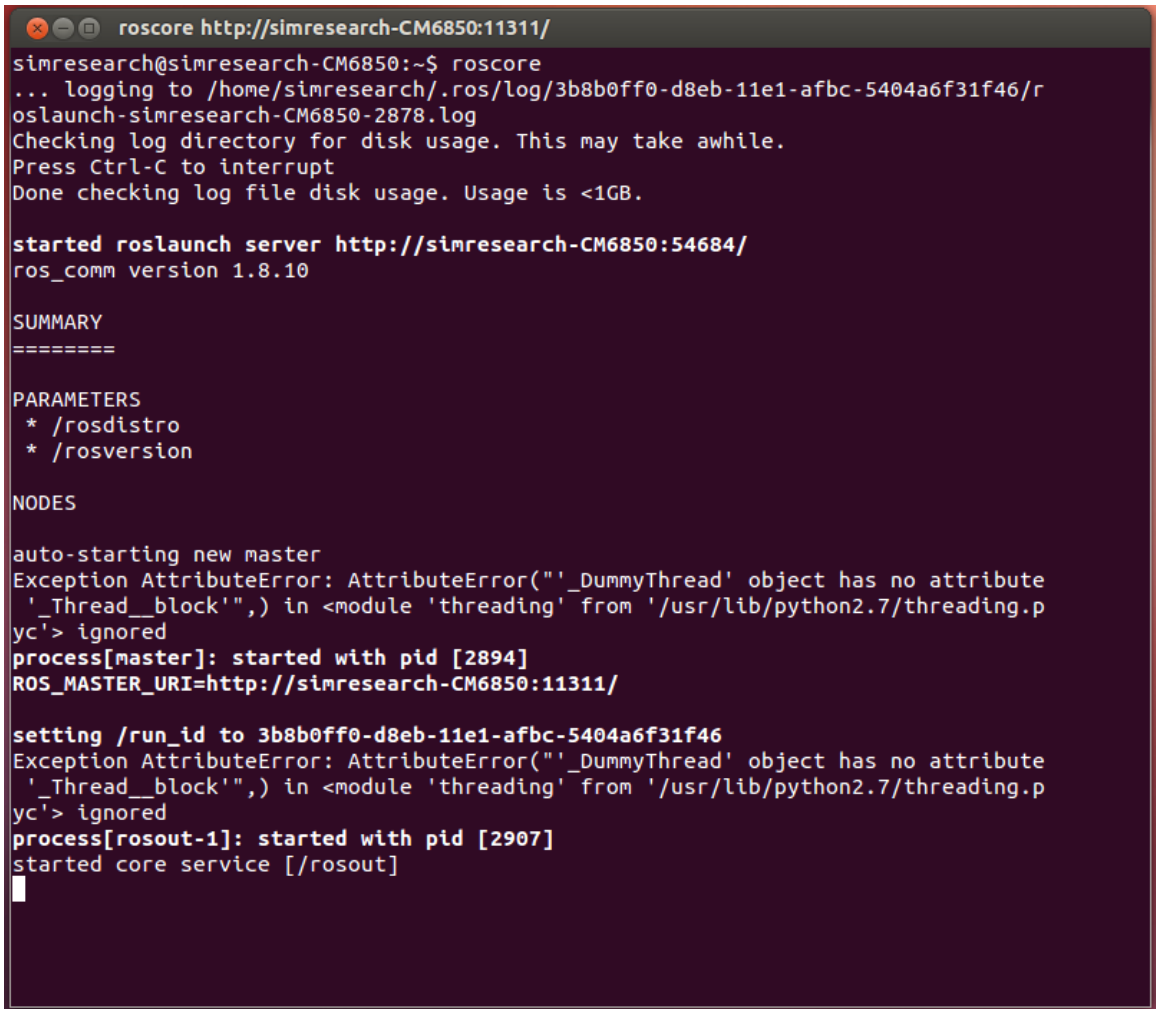
Importing Collada Mesh into ROS/Gazebo Environment

4.1 Opening an Empty Gazebo World

At this step, the file processing is complete and the road scan is ready to be opened in Gazebo. For this example, an Ubuntu machine running ROS Fuerte with the built in Gazebo package was used. The first step is to open a terminal and begin running the roscore using the command:

```
roscore
```

This is depicted in Figure 4-1.



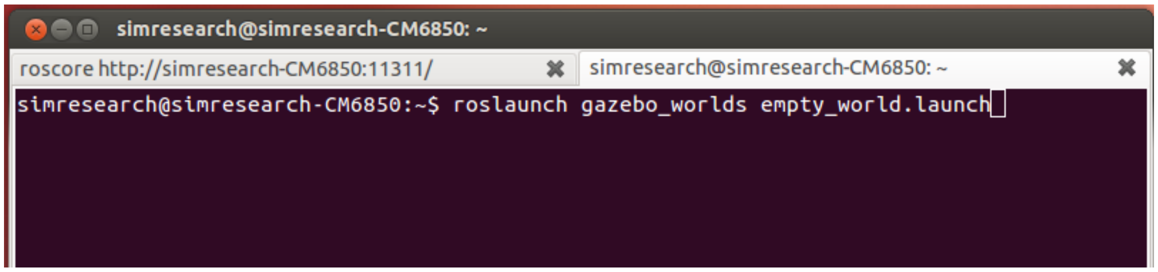
```
roscore http://simresearch-CM6850:11311/
simresearch@simresearch-CM6850:~$ roscore
... logging to /home/simresearch/.ros/log/3b8b0ff0-d8eb-11e1-afbc-5404a6f31f46/r
oslaunch-simresearch-CM6850-2878.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://simresearch-CM6850:54684/
ros_comm version 1.8.10

SUMMARY
========

PARAMETERS
 * /rosdistro
 * /rosversion

NODES

auto-starting new master
Exception AttributeError: AttributeError("'_DummyThread' object has no attribute
 '_Thread__block'",) in <module 'threading' from '/usr/lib/python2.7/threading.p
yc'> ignored
process[master]: started with pid [2894]
ROS_MASTER_URI=http://simresearch-CM6850:11311/

setting /run_id to 3b8b0ff0-d8eb-11e1-afbc-5404a6f31f46
Exception AttributeError: AttributeError("'_DummyThread' object has no attribute
 '_Thread__block'",) in <module 'threading' from '/usr/lib/python2.7/threading.p
yc'> ignored
process[rosout-1]: started with pid [2907]
started core service [/rosout]
```

Figure 4-1: Running the roscore

Open a new terminal tab by typing **Control + Shift + T**. In this terminal, run the command (Figure 4-2):

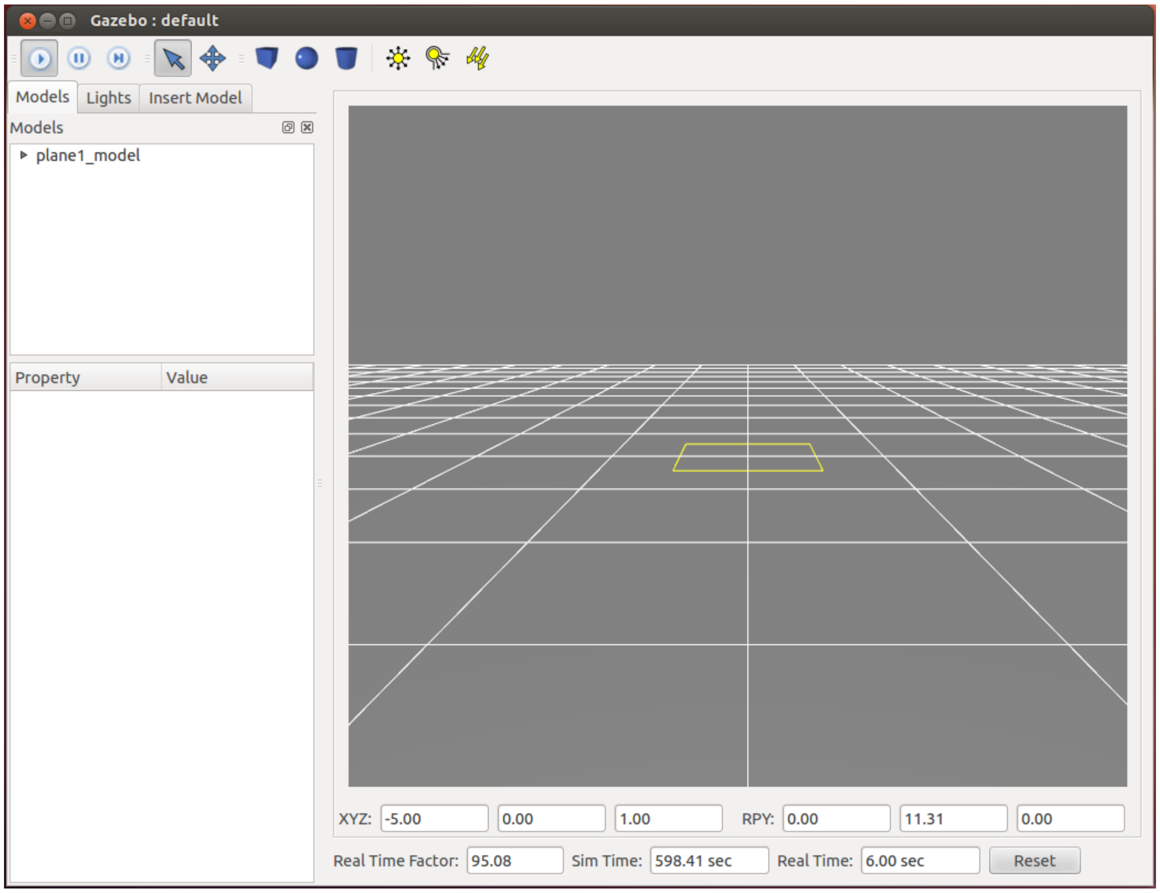
roslaunch gazebo_worlds empty_world.launch

This command will load an instance of Gazebo's default empty world, which can be seen in Figure 4-3.



```
simresearch@simresearch-CM6850: ~
roscore http://simresearch-CM6850:11311/    simresearch@simresearch-CM6850: ~
simresearch@simresearch-CM6850:~$ roslaunch gazebo_worlds empty_world.launch
```

Figure 4-2: Loading the empty world in Gazebo

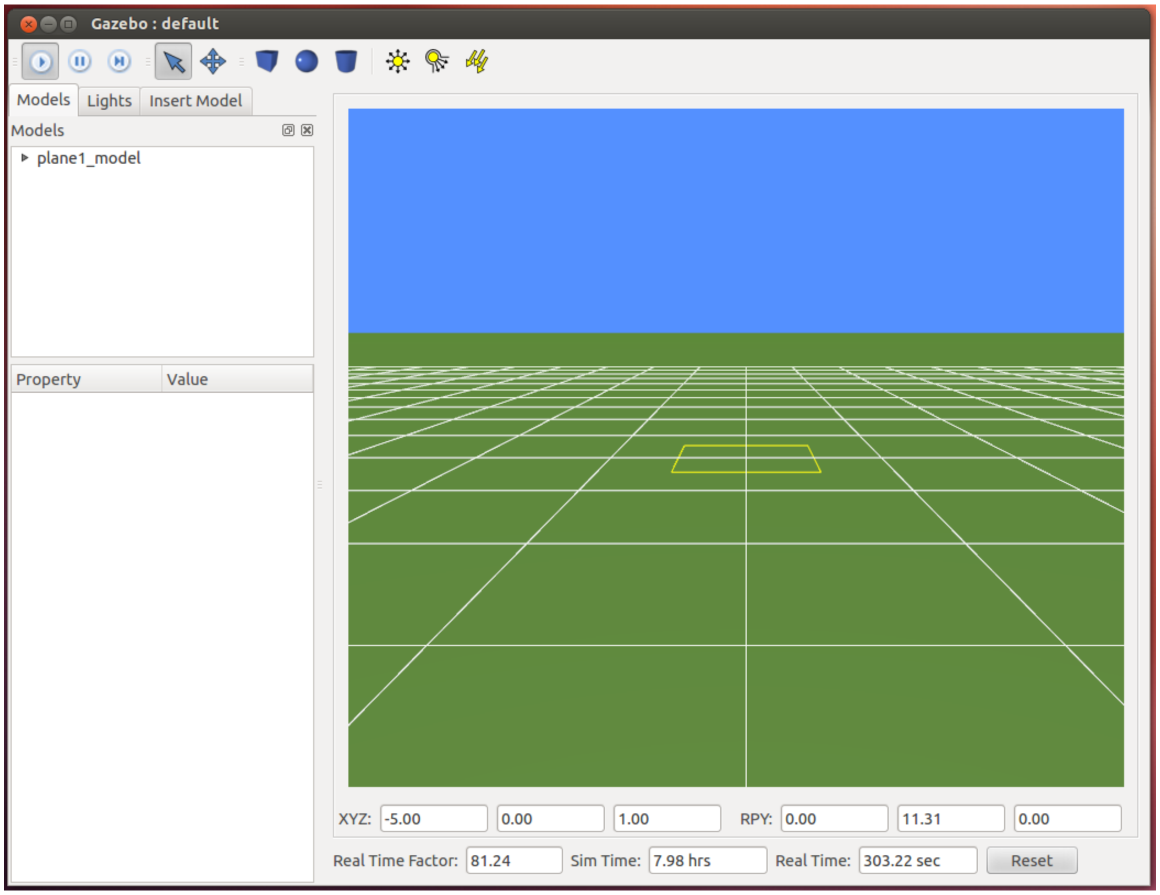


XYZ: -5.00 0.00 1.00 RPY: 0.00 11.31 0.00

Real Time Factor: 95.08 Sim Time: 598.41 sec Real Time: 6.00 sec Reset

Figure 4-3: The default empty world in Gazebo

To make the graphical display of the environment friendlier, the empty world can be modified by changing the fog and background settings. These improvements can be seen in Figure 4-4.



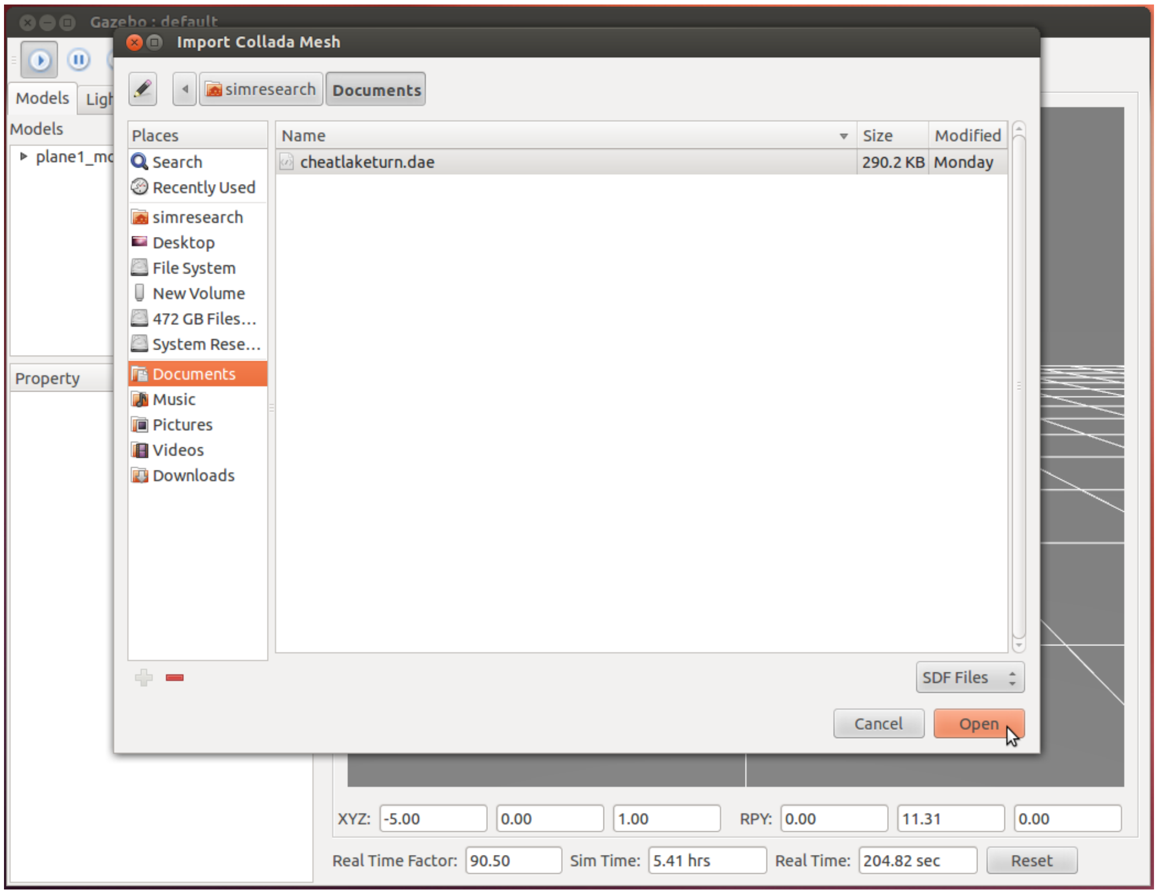
XYZ: -5.00 0.00 1.00 RPY: 0.00 11.31 0.00

Real Time Factor: 81.24 Sim Time: 7.98 hrs Real Time: 303.22 sec

Figure 4-4: More lifelike empty world in Gazebo

4.2 Importing Road Scan from Collada Mesh

To import the road scan, simply navigate to “File, Import Mesh” (Figure 4-5). Select the desired Collada Mesh (**cheatlaketurn.dae**), and select Open.



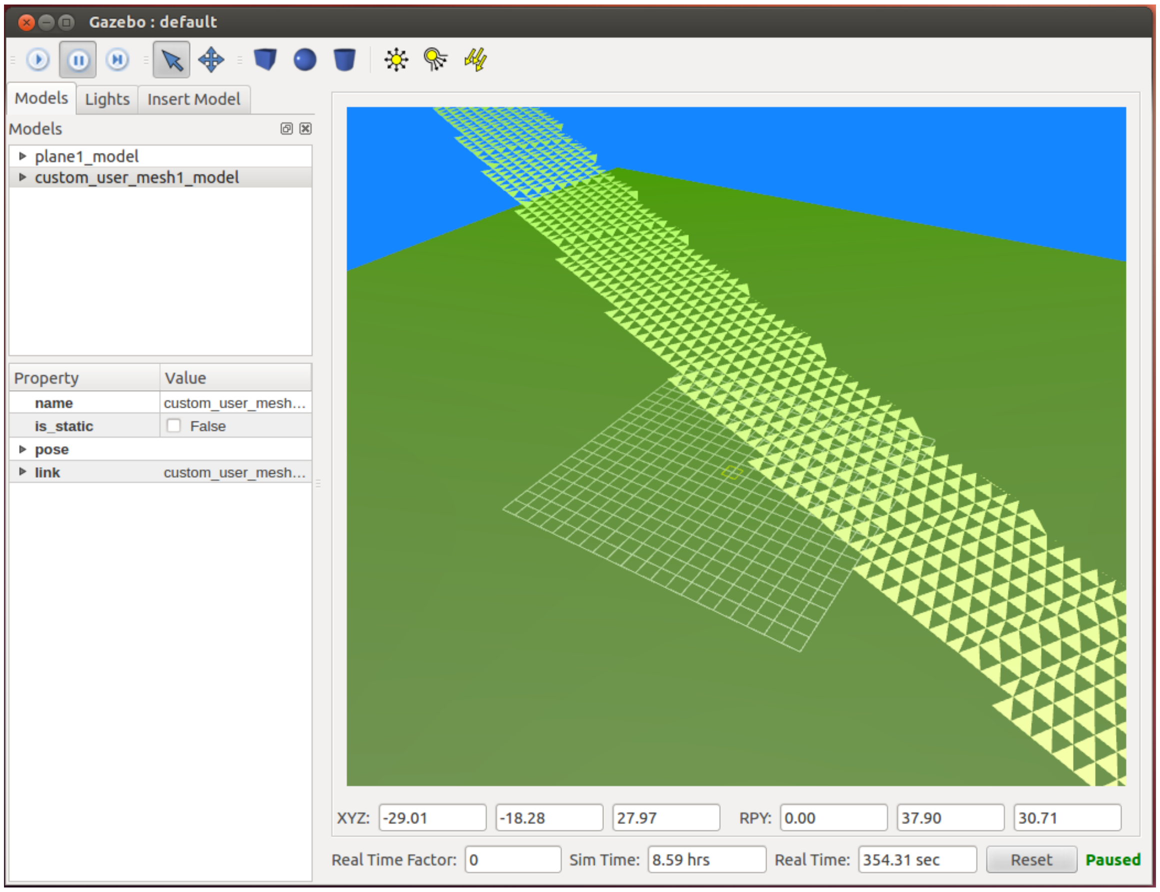
Name	Size	Modified
cheatlaketurn.dae	290.2 KB	Monday

XYZ: -5.00 0.00 1.00 RPY: 0.00 11.31 0.00

Real Time Factor: 90.50 Sim Time: 5.41 hrs Real Time: 204.82 sec

Figure 4-5: Importing the Collada Mesh into Gazebo

The road is loaded into Gazebo as shown in Figure 4-6 and again as a full screen simulation in Figure 4-7. At this point, further properties could be assigned to the road, such as color and texture, friction, gravity, and other visual and collision parameters.



Property	Value
name	custom_user_mesh...
is_static	False
pose	
link	custom_user_mesh...

XYZ: -29.01 -18.28 27.97 RPY: 0.00 37.90 30.71

Real Time Factor: 0 Sim Time: 8.59 hrs Real Time: 354.31 sec

Figure 4-6: Simulated Cheat Lake road in Gazebo

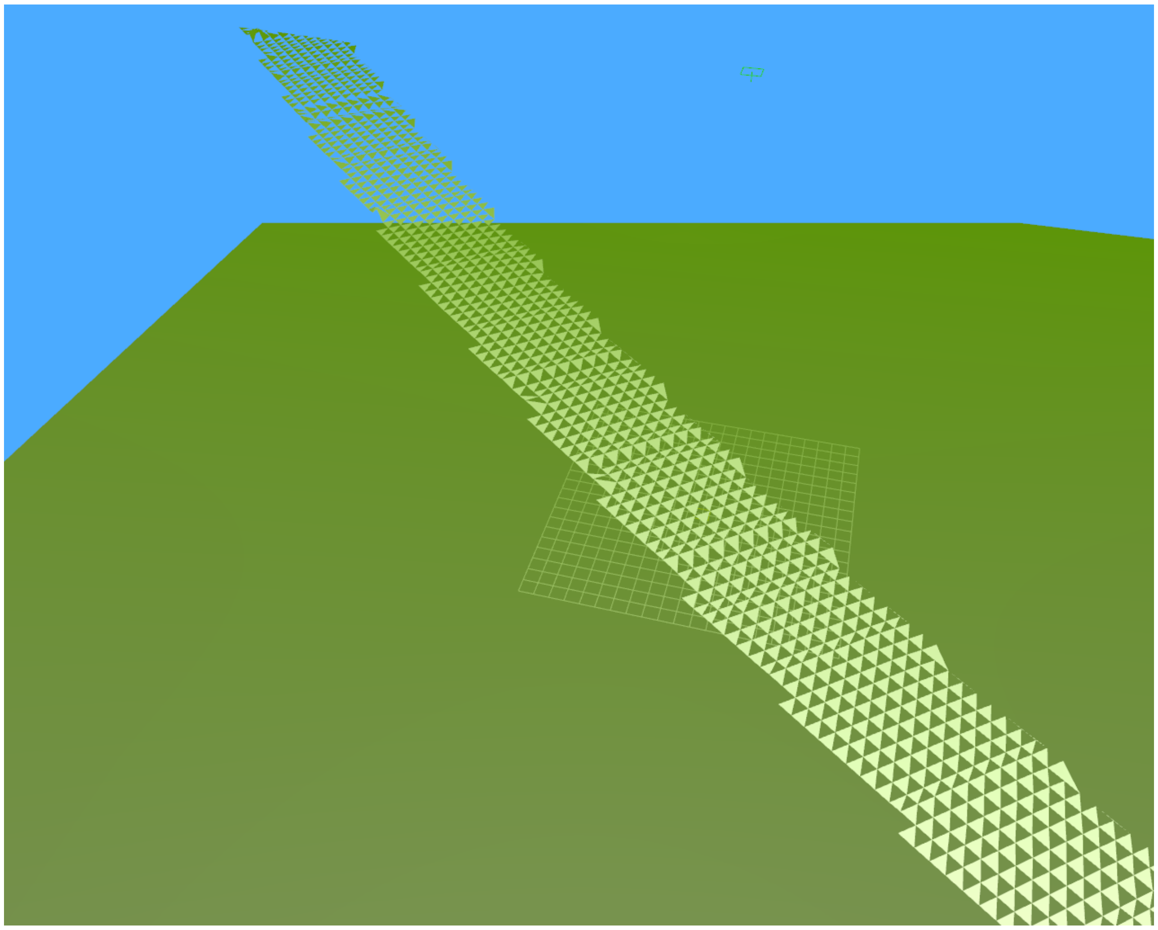


Figure 4-7: Full screen simulation of the Cheat Lake LIDAR scan using Gazebo Simulator

Chapter 5

Discussion of Results

5.1 Summary and Conclusions

Simulation (and more specifically automotive vehicle simulation) is an area of significant interest for both the research and industrial sectors. It has been shown that all areas of this science are receiving attention from researchers, including improving the lifelike characteristics of the simulators, studying human behavior using the simulator as a tool, and using the simulator to develop smarter more realistic autonomous vehicles. In an effort to restore Penn State's currently defunct driving simulator to working order, an attempt at rebuilding the entire simulator architecture using open-sourced free software is being investigated. This approach began from scratch, with the ultimate goal of producing a driving simulator of significant ability. The research team is hoping that one day, the system will be able to receive sensor outputs such as LIDAR and GPS data from a remote vehicle and use this data to construct a live simulation model in the cabin of the simulator. In addition, the hope is to then drive through the environment with the simulator, while the remote vehicle mimics the simulator's path, updating the environment sensors in real-time. The first step in this lofty project was to determine which software packages were best suited to handling this task. After significant research into the Robot Operating System (ROS) and Gazebo Simulator, it was determined that the combination of these two open-source software packages was not only capable of accomplishing this task, but in fact seemingly designed with these applications in mind. After establishing this, it was a second goal of this thesis to discover a way to convert sensor output (in this case LIDAR data) into a format that could be understood and used by the Gazebo simulator. This was accomplished using a few more open-source tools including Blender and a Python plug-in script. Using a LIDAR scan of Cheat Lake, WV, the point cloud data was ultimately skinned and converted into a Collada Mesh, which was successfully imported into the Gazebo simulator. Several other alternative methods were investigated, but all failed to achieve the same results as the Blender method. A third goal of this thesis was to take the simulated roadway and apply realistic properties using Gazebo such as color and texture, friction, and gravity. However, this goal was not met, as several different attempts to do so ultimately caused the Gazebo simulator to crash. The research ended with the failure to meet this goal.

Based upon the results of this research, several conclusions were drawn as to the effectiveness of this method. First, it is highly recommended to continue with the use of ROS and Gazebo Simulator as the software architecture of choice for this simulation effort. Their combined suitability for this research application was independently confirmed several times throughout the course of the literature review. Second, while the Blender method of processing the point cloud data was effective, it was inefficient and in its current form could not be used for running a continuously updating live simulator. Although a script could be written to automate this process, further research into ROS and Gazebo revealed that these two tools alone are capable of converting LIDAR data into a simulated environment, though a specific and detailed method to do so has yet to be discovered by this research. In addition, the Collada Mesh generated by this Blender method seemed to cause significant problems in Gazebo, resulting in the failure to implement lifelike properties on the model. It is because of these last two observations that the ultimate conclusion drawn from this method is that it should not be recommended for further study.

5.2 Suggestions for Future Research

As this thesis was essentially an introduction into a much bigger project, there are several suggested areas for future research. As previously mentioned, it is strongly suggested to pursue an alternative method than the one using Blender that was presented in this paper. Based upon further research, it is recommended to pursue a method that uses only ROS and Gazebo for this purpose. Such a method would require significant expertise in both ROS and the Gazebo Simulator, which was not achieved in this introductory research.

In addition to the topics covered in this thesis, it would also be necessary to determine a way to utilize the Gazebo software on a multi-screen simulator system. The driving simulator at Penn State currently has three screens simulating vision out of the front windshield, and also one additional screen on each side to represent views seen in the side mirrors. Because all of Gazebo's key features can be run headless (without a GUI), it might be necessary to implement some other kind of graphics engine to create the multiple displays that would be required.

Other research topics that would be necessary to explore but differ significantly from the scope of this thesis include determining ways to send data to and from the remote vehicle and the driving simulator software, wiring the gas pedal, brake pedal and steering wheel of the simulator to serve as inputs to the remote vehicle, and creating a cabin platform that could simulate acceleration in 3D using roll, pitch and liner lateral motion. The completion of this project would take several months if not years, and require significant research into several different fields of research above and beyond the virtual worlds explored in this thesis.

Appendix A

Results of Institute for Psychosocial Medicine Study on Sleepless Driving

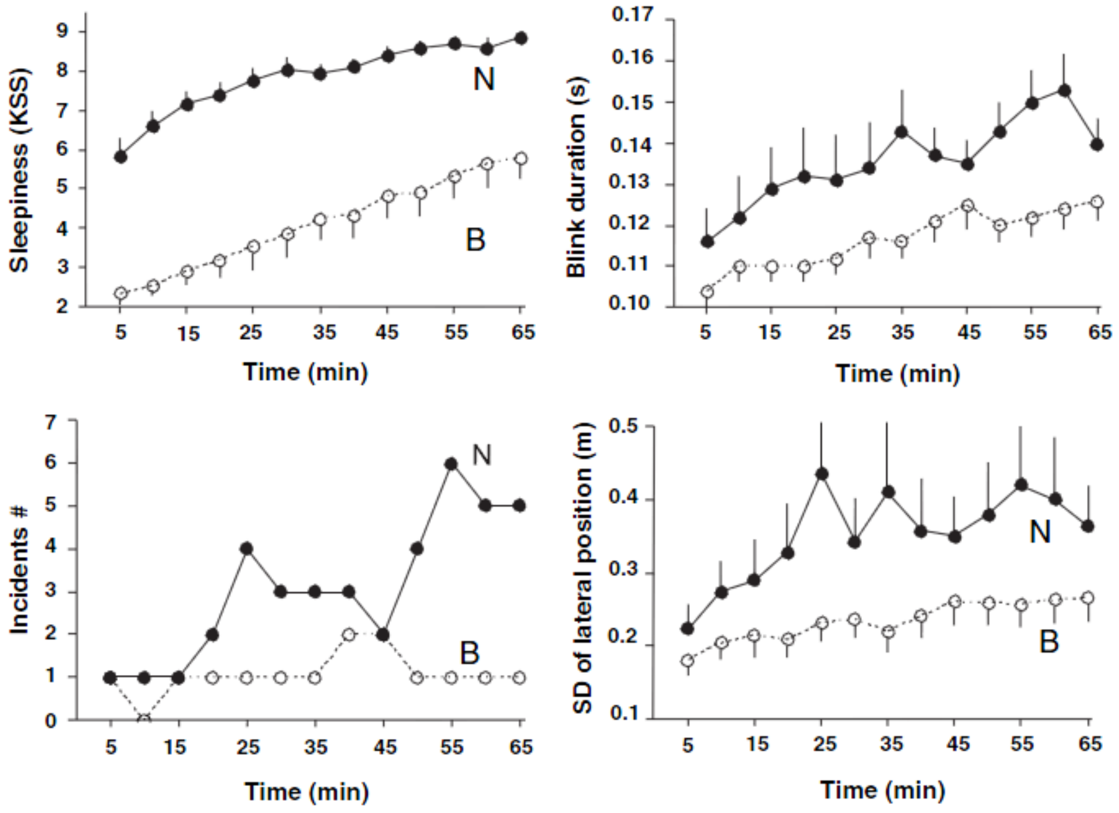


Figure A-1: Results of four different measures of driving performance. N = no sleep/night work condition. B = baseline condition (adopted from Åkerstedt et al., 2004)

Appendix B

Results of Penn State Study on 3D Terrain Modeling

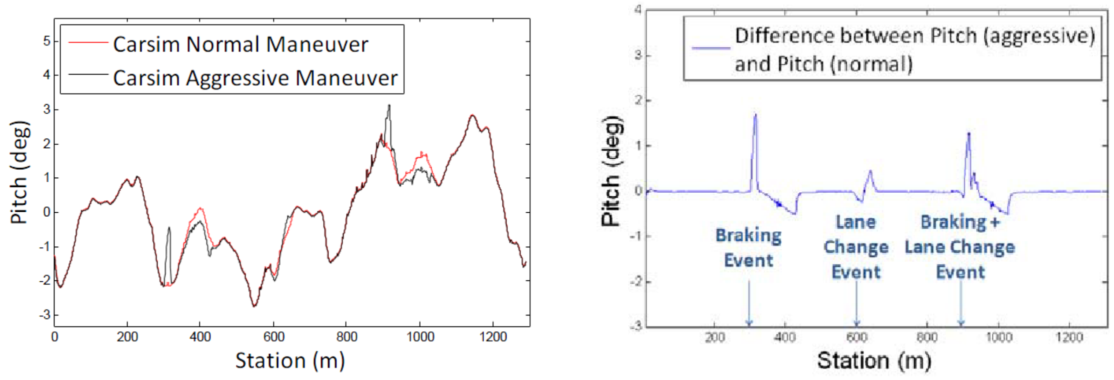


Figure B-1: Emergency maneuver detection based on pitch data (adopted from Varunjikar et al., 2011)

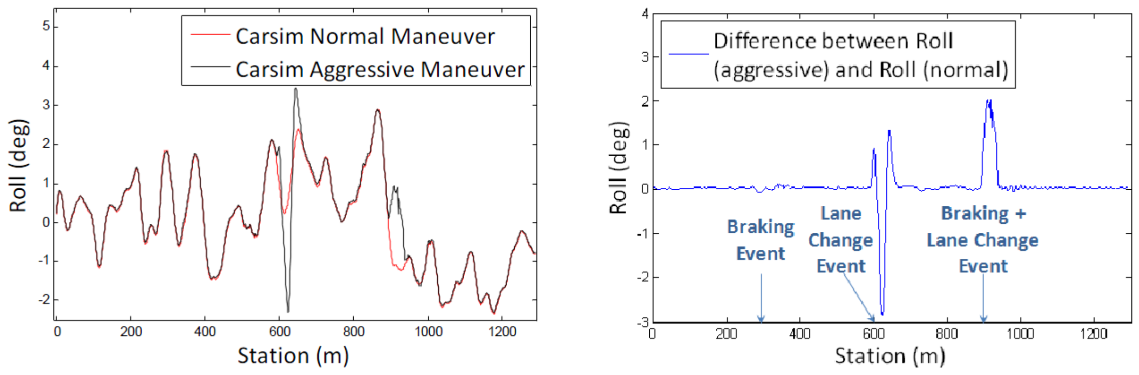


Figure B-2: Emergency maneuver detection based on roll data (adopted from Varunjikar et al., 2011)

References

"About Gazebo." *GazeboSim*. Web. 20 July 2012. <http://www.gazebosim.org/about.html>.

Akerstedt, Torbjorn, Bjorn Peters, Anna Anund, and Goran Kecklund. *Impaired Alertness and Performance Driving Home from the Night Shift: A Driving Simulator Study*. Tech. European Sleep Research Society, 22 Oct. 2004. Web. 17 July 2012.

Al-Shihabi, Talal, and Ronald R. Mourant. "A Framework for Modeling Human-like Driving Behaviors For Autonomous Vehicles in Driving Simulators." Proc. of International Conference on Autonomous Agents. Northeastern University, June 2001. Web. 20 July 2012.

Beckman, Wendy S. "Pilot Perspective on the Microsoft Flight Simulator for Instrument Training and Proficiency." *International Journal of Applied Aviation Studies* 9.2 (2009): 171-80. Web. 20 July 2012.

Bouchard, Samuel. "Robot Operating System Making Its Way Into Industrial Robotics." *Ieee Spectrum*. IEEE, 21 Sept. 2011. Web. 16 July 2012. <http://spectrum.ieee.org/automaton/robotics/robotics-software/robot-operating-system-making-its-way-into-industrial-robotics>.

Diewald, Stefan. "Implementation of a Development Toolchain for the Simulation of Mobile Devices Using the ROS Middleware." Thesis. Technische Universitat

Munchen, 2011. 27 May 2011. Web. 20 July 2012. <https://vmi.lmt.ei.tum.de/publications/students/Diewald-Diplomarbeit.pdf>.

Download. Blender, n.d. Web. 2 July 2012. <http://www.blender.org/index.php?id=1262&f=http://download.blender.org/release/Blender2.63/blender-2.63a-linux-glibc27-x86_64.tar.bz2>.

Guizzo, Erico. "Microsoft Shifts Robotics Strategy, Makes Robotics Studio Available Free." *Ieee Spectrum*. IEEE, 20 May 2010. Web. 20 July 2012. <http://spectrum.ieee.org/automaton/robotics/robotics-software/052010-microsoft-shifts-robotics-strategy-makes-robotics-studio-available-free>.

Hans.P.G. "A Script to Skin a Point Cloud (for Blender 2.6x or Later)." BlenderArtists.org, 8 Jan. 2012. Web. 28 June 2012. <http://blenderartists.org/forum/showthread.php?241950-A-Script-to-Skin-a-Point-Cloud-%28for-Blender-2.6x-or-Later%29>.

Koenig, Nathan, and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator." Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan. University of Southern California, Sept.-Oct. 2004. Web. 19 July 2012. <http://cres.usc.edu/pubdb_html/files_upload/394.pdf>.

Lucas, Simon M., and Renzo De Nardi. "Computational Intelligence in Racing Games." *Advanced Intelligent Paradigms in Computer Games*. By Julian Togelius. N.p.: Springer Berlin / Heidelberg, 2007. 39-69. Web. 12 July 2012. <http://www0.cs.ucl.ac.uk/staff/R.DeNardi/Togelius2007Computational.pdf>.

Pickem, Daniel, David Morioniti, Chris Taylor, Santiago Balestrini-Robinson, and Dimitri Mavris. *Captain Hindsight: An Autonomous Surface Vessel*. Tech. Georgia Institue of Technology, Aug. 2011. Web. 17 July 2012. <http://mrg.gatech.edu/wordpress/wp-content/uploads/2011/08/paper_reduced.pdf>.

Plagemann, Christian, Cyrill Stachnis, and Wolfram Burgard. *Efficient Failure Detection for Mobile Robots Using Mixed-Abstraction Particle Filters*. Tech. University of Freiburg, 2006. Web. 20 July 2012. <http://www.informatik.uni-freiburg.de/~plagem/bib/plagemann06euros.pdf>.

Quigley, Morgan, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. "ROS: An Open-source Robot Operating System." Diss. Stanford University, 2009. Stanford University. Web. 20 July 2012. <http://ai.stanford.edu/~ang/papers/icraoss09-ROS.pdf>.

Rekleitis, Ioannis, Jean-Luc Bedwani, and Erick Dupuis. "Autonomous Planetary Exploration Using LIDAR Data." Proc. of 2009 IEEE International Conference on Robotics and Automation, Kobe International Conference Center, Kobe, Japan. IEEE, May 2009. Web. 19 July 2012. <http://www.cim.mcgill.ca/~yiannis/Publications/ICRA_2009a.pdf>.

Remo, Chris. "Microsoft Makes Big Cuts At Flight Sim Studio." *Gamasutra*. N.p., 23 Jan. 2009. Web. 12 July 2012. <Microsoft Makes Big Cuts At Flight Sim Studio>.

Robertson, Adi. "Epic Licensing Unreal Engine 3 for FBI Training Sim and Other 'serious Games'" *The Verge*. N.p., 28 Mar. 2012. Web. 16 July 2012.

<http://www.theverge.com/2012/3/28/2908051/epic-unreal-engine-3-licensed-government-agencies>.

"RoboBoat Overview." AUVSI Foundation, n.d. Web. 20 July 2012. <http://www.auvsifoundation.org/foundation/competitions/roboboat/roboboatoverview/>.

"ROS." *Willow Garage*. N.p., n.d. Web. 20 July 2012. <http://www.willowgarage.com/pages/software/ros-platform>.

Rusu, Radu B., Alexis Maldonado, Michael Beetz, and Brian Gerkey. *Extending Player/Stage/Gazebo towards Cognitive Robots Acting in Ubiquitous Sensor-equipped Environments*. Tech. Technische Universitat Munchen, 2007. Web. 19 July 2012. <http://www.willowgarage.com/sites/default/files/Rusu07ICRA_NRS.pdf>.

Sarkar, Samit. "'NHL 13' Challenges You to Be a Smarter Hockey Player." *The Verge*. N.p., 4 July 2012. Web. 20 July 2012. <http://www.theverge.com/gaming/2012/7/4/3135416/nhl-13-challenges-you-to-be-a-smarter-hockey-player>.

Smith, Russell. "Open Dynamics Engine V0.5 User Guide." N.p., 23 Feb. 2006. Web. 18 July 2012. <https://www.mmpong.net/trac/export/b3244fd9c50885e050e826f0916bd03d8df9a51b/doc/ode-latest-userguide.pdf>.

Varunjikar, Tejas M., Pramod K. Vemulapalli, and Sean N. Brennan. *Multi-Body Vehicle Dynamics Simulation Based On Measured 3D Terrain Data*. Tech. Pennsylvania

State University Department of Mechanical & Nuclear Engineering, n.d. Web. 16 July 2012.

Volvo Group. R&D. *Sweden's Most Advanced Driving Simulator Comes to Gothenburg*. *Volvo Group Global*. N.p., 18 May 2011. Web. 16 July 2012. <http://www.volvogroup.com/group/global/en-gb/researchandtechnology/news_updates/_layouts/CWP.Internet.VolvoCom/NewsItem.aspx?News.ItemId=102523&News.Language=en-gb>.

Academic Vita of Nicholas R. Sirera

Nicholas R. Sirera

315 West Beaver Avenue	nrs5054@psu.edu
Apartment 8	
State College, PA 16801	

Education

The Pennsylvania State University	B.S. Mechanical Engineering
Schreyer Honors College	B.S. Nuclear Engineering
College of Engineering	Engineering Entrepreneurship Minor
University Park, PA	

Thesis Title: Preparation of LIDAR Point Cloud Data for Use in a ROS/Gazebo Simulation Environment

Thesis Supervisor: Dr. Sean Brennan

Work Experience

PSU Intelligent Vehicles and Systems Group	(May 2011 – August 2012)
Undergraduate Researcher	
Supervisors: Dr. Sean Brennan, Alex Brown	

Curtiss-Wright Corporation	(January 2009 – August 2010)
Engineering Co-op	
Supervisors: Brian Eckels, Ross Klein, Paul Stasko	

Awards

Spark's Award (Recognition of 4.0 GPA after sophomore year)
President's Freshman Award (Recognition of 4.0 GPA after freshmen year)
Collegiate Roller Hockey Scholar Athlete of the Year (2011-2012 Season)
Collegiate Roller Hockey National Champions (2009-2010 Season)
William J. and Lois Kesterson Leight Memorial Scholarship
Professor H. A. Everett Memorial Scholarship
Louis A. Harding Memorial Scholarship
Nicholas A. Petrick Scholarship
Arthur Wilcox Memorial Award
Exelon Nuclear's Contributions Program Candidate
Valedictorian, Pine-Richland High School Class of 2007

Activities

Penn State University Roller Hockey Captain/Member