Autonomous Driving Simulator for Self-Driving Vehicle Development and Familiarization

A **Technical Report** Submitted to the

Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements of the Degree

Mechanical Engineering, Bachelor of Science, School of Engineering

John Grant

Spring 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Signature John M. Grant Date 11 Mar 2022

John Grant

Signature [illegible] Date 5/11/2022

Technical Advisor, Tomonari Furukawa, Department of Mechanical Engineering

Part I - Introduction

There is no doubt that the integration of autonomy in modern technologies is a rampant trend among almost every sector of the world's industries today. Autonomous systems have been implemented in manufacturing to ease the burden of menial and potentially dangerous tasks, in the commercial world to disrupt human-to-human interactions in light of a global pandemic, and most relevantly to this topic, in the automotive industry to free our ever-busy hands and minds while commuting in cars. Surely autonomy is prevalent in a plethora of other fields, but it is in these sectors where ethical dilemmas sprout from underneath every effort at its implementation into technology. As human oversight is disconnected from the functions of these technologies, there arises concern in how they will behave in situations which commonly require the complicated nature of human intuition and decision-making. In general, the dilemma our society faces as autonomy spreads throughout the world is how it will be safely integrated into organizations and human daily life. There are many ways to navigate the process of fostering public trust in autonomous systems, but the particular one of interest in this context is simulated studies. Specifically, virtually simulating autonomous automobiles and how they interact with their environment. Simulation software such as this can allow self-driving vehicle manufacturers to gain insight on how the algorithms their autonomous systems utilize hold up in potential real-world scenarios. While manufacturers can alternatively just test their technology in the real-world, this often incurs undesirable risks and consequences, such as harm to vehicle operators, pedestrians, and other third parties. An autonomous driving simulator would give these companies a platform in which they can safely test and develop their systems. Additionally, simulators can also serve to familiarize consumers with operating autonomous vehicles. As self-driving vehicles become more prevalent throughout the world, it's important that the common user feels comfortable riding in an autonomous car. Experiencing it first in a simulated environment would be an effective way to ease consumers into familiarity with this technology. I am in a mechanical engineering capstone group where we are developing our own autonomous driving simulator. With these concerns in mind, we aim to develop an autonomous driving simulator as a valuable resource for self-driving vehicle production and familiarization.

Part II – Literature Review

Pre-existing research into the development of autonomous driving simulators shows that similar approaches to our problem have been attempted, but none fully address the goals we seek to accomplish. Nonetheless, these serve as valuable examples by which we can guide our own research and development. Each of the referenced studies can be classified into one of two approaches: a primary focus on the user experience and immersivity or a primary focus on external factors and functions, such as simulated traffic or pedestrians. With respect to the first kind of approach, a 2015 study conducted by Baltodano, et. al. documented the development of the Real Roads Autonomous Driving Simulator (RRADS). Their research aimed to gather the overall public attitudes and expectations of how an autonomous vehicle is intended to feel. The data was then used to inform the ongoing development of their simulator, and specifically to maximize the immersivity of their design. Other studies that have sought to address immersivity and user experience include Koo et. al. (2014), which was concerned with user understanding of safety features; Bellem et. al. (2016), which sought to assess whether or not moving-base driving simulators induce a comfortable and realistic sense of motion; and Häuslschmid et. al. (2017), where it was found that indicator displays showing the autonomous algorithm actively processing external data fostered greater user trust. The findings of these studies serve to inform our own development, as we can gauge what consumers seek in a simulated autonomous experience.

Several other studies align more with the second approach we've established, where there's a greater focus on external functions in the simulated environment. In 2017, You et. al. published a paper that detailed their research into generating realistic virtual environments for autonomous driving simulators. They developed a system which took non-realistic virtual image input and processed it into realistic scene structures. Similar studies include Chao et. al. (2020), which documented research into reconstructing detailed traffic flows using real-time traffic data with traffic simulation models; and Pérez-Gil et. al. (2022), where the use of deep learning algorithms was proposed as a means to create more effective simulated autonomous algorithms.

Part III – Methodology & Design Process

Before delving into the specific breakdown of autonomous simulators, it is important to establish an understanding of basic driving simulators. There are several components which make up the technology of a driving simulator. The two main pieces are the simulation software and the physical controls/chassis. These two elements are interfaced with one another through electronic/mechatronic systems, which constitute sensors, actuators, etc. The software, mechatronic systems, and physical user-controlled components all coalesce to create an immersive, simulated driving experience.

In the software sector, there are several programs which work together to generate the computations and visual and haptic feedback of the simulator experience. Certain software is required to compute the physics which characterize the behavior of the virtual environment, while other software is used to generate the graphics, although it is common for some programs to integrate both of these functions. Additionally, there are programs required to relay the digital data to the actuators and sensors in the physical driving simulator. For a software developer working on a simulator like this, much of their job is concerned with efficiently interfacing all the separate programs in order to reduce latency and computational energy. Therefore, there is a critical balance to be achieved between the complexity and efficiency of simulator software. My capstone group has identified this as a key factor in the development of our own simulator. Based on the feedback of our potential customers, one of the primary goals of our software team is to minimize latency between user input and the graphical response. This will involve utilizing our core software, Robot Operating System (ROS), to interface our more specialized programs. Among these are Gazebo, a virtual environment generator; and OpenDS, a physics simulation program. ROS essentially receives the outputs of these two programs and merges them to render the complete virtual environment. Additionally, it will output all the necessary data to the user display and physical actuators. Our target specification regarding latency between user input and graphical/haptic feedback is 10-15 milliseconds. Therefore, our goal is to create the most realistic virtual environment while maintaining this specified constraint on latency. There are a few methods by which this could be achieved. One involves removing OpenDS from the system architecture and utilizing Gazebo for both the physics and virtual environment generation. In this way, we would reduce some of the latency contributed from the cross-platform communication between the two separate programs. However, this would require more system memory to be allocated to Gazebo and could potentially increase its computational time. Our group decided to go with a different option, which was to use a software called CARLA for all aspects of the environment generation. CARLA computes the physics, generates the visualizations, and includes autonomous sensor simulation options. This proved to be the most effective way to mitigate latency, as all functions are consolidated into a single program.

The chassis of a driving simulator is made up of all the user controls and structural components. The physical sector's main purpose is to foster user immersion in the simulator experience (Meywerk, 2016). Components include controls such as the driving wheel, pedals, gear shift stick, etc. It is common for simulators to implement the actual hardware used in real cars to maximize immersion. The chassis would also have things such as the seat, dashboard, and visual display. All the user controls are interfaced with the simulation software using sensors, encoders, potentiometers, and more to translate the mechanical motion of each component into digital data to be processed in the virtual realm. Haptic and visual feedback are provided to the user by means of actuators, vibrational emitters, and electronic displays. As the car virtually encounters features such as bumps, hills, and turns, these elements would manipulate the driver's cabin to induce an appropriate sense of motion. Our team's goal for this section of our project is to prioritize user immersion. Likewise, we have installed the chassis of a Subaru Forester to provide the most realistic user interface, which is shown in Figure 1 below. Additionally, we have installed a surrounding projector display system as per Figure 2.

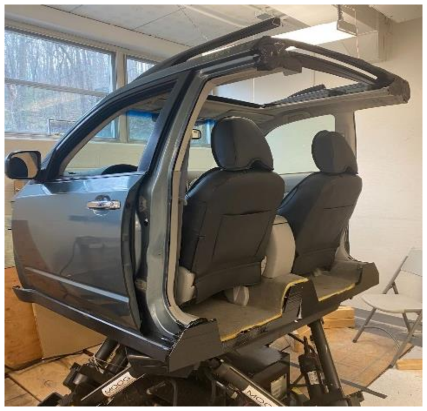
Figure 1: Simulator chassis

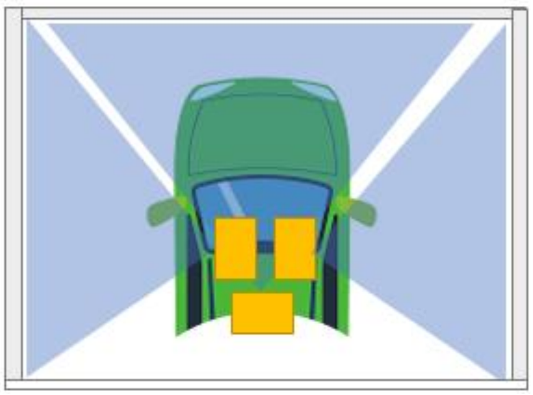
Figure 2: Three-projector display system

An autonomous driving simulator includes all of the aforementioned components, though the main differences exist in its software. Programs capable of simulating and integrating autonomous algorithms are required to achieve simulation of self-driving cars. Some of these include NVIDIA DriveSim, CARLA, SUMMIT, and more. As autonomous vehicle production has increased, there's been a substantially increasing trend in the development of software such as these to simulate their functionality. For our particular project, we aim to use RTMaps, a lower-level autonomous sensor simulation program. The previously mentioned software all include physics computations and environment generation, while RTMaps is solely for simulating self-driving vehicle sensors and testing autonomous algorithms. This will allow us more freedom to modify our software architecture in order to achieve our target specifications.

From a technical perspective, one of the main goals a self-driving car simulator seeks to accomplish is an accurate rendering of real-world scenarios. This is the core of the whole purpose of the software: to virtually insert an autonomous vehicle and its algorithms into a potential real-world scenario. The software manages this by generating realistic environments for the virtual vehicle to navigate and simulating the characteristics of the autonomous algorithms to a certain degree of precision. Different simulators achieve this in various ways. The CARLA driving simulator seeks to assess the performance of three different approaches in autonomous driving: a classic modular pipeline, an end-to-end model trained via imitation learning, and an end-to-end model trained via reinforcement learning (Dosovitskiy et al., 2017). In this case, users of this software must adapt their algorithms to one of those three methods of autonomous development. On the other hand, a software such as Nvidia DriveSim supports a larger-scale, more complex simulation of the self-driving car sensors (Mirocha, 2018). For our own simulator, we have implemented CARLA as our virtual environment software. This software comes packaged with all the physics processing, environment rendering, and autonomous sensor simulation options that we require to synthesize our design.