Designing a Deep Learning and Computer-Vision Based Autonomous Vehicle Within a Multimodal Traffic Simulation Framework

Vijay Kalivarapu, Adam Kohl, Jack Miller, Eliot Winer
Iowa State University
Ames, IA
vkk2@iastate.edu, adamkohl@iastate.edu, jackm@iastate.edu, ewiner@iastate.edu

ABSTRACT

In 2014, the industry had invested approximately $167 million into Autonomous Vehicle (AV) technologies and research. By 2019, these investments totaled more than $100 billion. The Pentagon's 2020 fiscal year budget proposal included $3.7 billion for research and development of unmanned and autonomous technologies. Studies show that 52% of battlefield casualties occur when soldiers deliver food and other supplies in combat zones. AVs offer the potential to mitigate such risks and save lives substantially. However, AVs must be tested in a multitude of scenarios before they are practically viable for military and civilian applications. Physical AV data for testing is generally unavailable from commercial or military entities due to proprietary or security concerns. This makes simulations a feasible alternative to study them. However, creating AV simulations with the fidelity, scalability, and customization comes with several research questions such as 1) How can AVs be trained for autonomous driving? 2) How can communication be established between different traffic management subsystems? 3) How can multi-user coordination and collaboration, in such an environment, be achieved?

A three-component visualization framework was developed to answer the above questions. First, multiple virtual autonomous vehicles were trained using machine learning techniques to drive within a specific road intersection scenario. Second, these virtual AVs were introduced to physical agents such as cars and bike riders. Third, the driving states of the physical agents and the AVs were synchronized using a client-server architecture with a traffic simulator that probabilistically generated vehicle and pedestrian traffic. The AVs and the physical agents appear as entities within the traffic simulator to which the generated traffic computes responses and are network synchronized to form a multimodal traffic simulation system collectively. Results from implementing and testing this framework in multiple scenarios show that adequately trained AVs can serve as a crucial first step, a proof-of-concept validation, for developing military and civilian AV applications.

ABOUT THE AUTHORS

Vijay Kalivarapu, Ph.D., is a staff research scientist at the Virtual Reality Applications Center, Iowa State University. Dr. Kalivarapu has extensive experience in Virtual Reality, 3D Computer Graphics, and Design Optimization.

Adam Kohl is a Ph.D. candidate in Mechanical Engineering Computer Engineering at Iowa State University's Virtual Reality Applications Center. His research interests include the development of pattern recognition techniques and supervised methods for engineering applications.

Jack Miller is a Ph.D. candidate in Computer Engineering and Human-Computer Interaction at the Iowa State University's Virtual Reality Applications Center. His current research interests include exploring multi-user experiences in virtual and augmented reality environments, specifically for traffic simulation.

Eliot Winer, Ph.D., is the director of the Virtual Reality Applications Center and professor of Mechanical Engineering, Electrical and Computer Engineering, and Aerospace Engineering at Iowa State University. Dr. Winer has over 20 years of experience working in virtual reality and 3D computer graphics technologies on sponsored projects for the Department of Defense, Air Force Office of Scientific Research, Department of the Army, National Science Foundation, Department of Agriculture, Boeing, and John Deere.

Designing a Deep Learning and Computer-Vision Based Autonomous Vehicle Within a Multimodal Traffic Simulation Framework

Vijay Kalivarapu, Adam Kohl, Jack Miller, Eliot Winer
Iowa State University
Ames, IA
vkk2@iastate.edu, adamkohl@iastate.edu, jackm@iastate.edu, ewiner@iastate.edu

INTRODUCTION

The Defense Advanced Research Projects Agency (DARPA)'s $1 million grand challenge in 2004 called for a driverless car competition to autonomously drive 132 miles of terrain in the Mojave Desert region within 10 hours (Behringer, R, 2004). While there was no winner to claim the prize, the team from Carnegie Mellon University traveled the furthest – 7.3 miles. This event jumpstarted the careers of many experts and companies, resulting in a $100 billion commercial Autonomous Vehicle (AV) industry by 2019. In February 2019 alone, there was a $1.6 billion investment in AV companies for civilian applications (Demaitre, E, 2019). The benefits of autonomous technologies for civilian applications are very tangible. Fewer traffic congestions with improved safety, decreased driver fatigue, lower emissions, improved health, and higher demand for new jobs are just some of the outcomes attributable to AV technologies in the civilian sector.

Further, there is a pronounced interest in developing AV technologies in the US military. The Pentagon's 2020 budget proposal of $3.7 billion in research and development of 'unmanned and autonomous technologies' serves as credible evidence that the US military is paying attention to these technologies for various battlespace applications (Muller, J, 2019). Reports and studies show that 52% of battlefield casualties occur when soldiers deliver food and supplies in conflict and combat zones (United States Senate Committee on armed services transcript, 2018). It was theorized that unmanned AVs with AI driving algorithms could substantially mitigate risks and save lives. However, detailed intelligence, including road maps in war zones, are not always readily available or can change within a short period of time. Additionally, road signs, if available, are prone to be damaged and can be misleading. These constraints, which typically do not exist in civilian applications, make the development of AV technologies for military applications a multifold challenge.

Regardless of civilian or military applications, the development of AV technologies requires substantial testing in a multitude of scenarios before they are practically viable. Testing AVs during their development within live traffic on US road systems is impractical due to safety reasons. Similarly, AV testing in combat zones can be chaotic and dangerous. However, simulations of AVs are repeatable, and conditions can be controlled to mimic real scenarios, making them a practical and safer alternative. The work presented in this paper demonstrates the design of an AV in a multimodal traffic simulation framework, with several generalized research questions addressed, such as:

- How can AVs be trained for autonomous driving in a simulated driving environment?
- How can communication be established between different traffic management subsystems?
- How can multi-user coordination and collaboration, in a simulated driving environment, be achieved?

The remainder of the paper is organized as follows: First, the background literature of companies and organizations developing civilian and military AVs is presented. Then, a list of commercial/open-source traffic simulator and AV simulator applications are summarized. A literature review of applications used for simulation and testing of AV algorithms will also be discussed, followed by a general procedure on how AVs are trained and integrated into traffic simulator programs. The methodology section will include the architecture designed and created for a collaborative client-server traffic simulation environment, including how AVs were trained and deployed within it. Two test cases for single and three-lane roads are then discussed, followed by results and conclusions.

BACKGROUND

Prior AV Work

Although AV technologies gained prominence in recent years, the research areas is not new. The first reported instance of an unmanned vehicle dates back to 1925 in New York, where Francis Houdina modified a 1926 model Chandler car equipped with a transmitter antenna, as seen in Figure 1 (Green, H., 1925). In 1957, the company RCA embedded detectors within a patch of 400-ft road on US 77 highway in Nebraska to send impulses and guide a Chevrolet car as it drove by (Hicks, N., 2017).

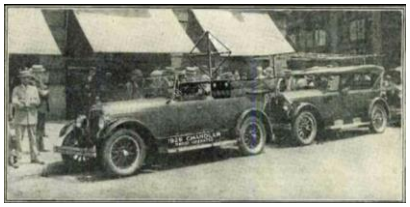


Figure 1. Driverless car from 1925

Figure 2. Autonomous Land Vehicle

The first AV to use cameras was the Autonomous Land Vehicle (ALV) from the US government's Strategic Computing Initiative in the 1980s. It was built by Martin Marietta (now merged to become Lockheed Martin) and the University of Maryland through DARPA funding, and is shown in Figure 2 (Leighty, R, 1986) (Waxman, A. 1987) (Lockheed Martin) (Paleofuture, 2007).

Although there were other lesser-known US federal government initiatives in the late 1980s and 1990s (Dhawan, C., 2019), breakthroughs in AV development did not occur until the 2000s. In addition to the DARPA grand challenge of 2004, which did not yield any winners, the 2005 challenge resulted in five winners that successfully were able to complete the autonomous driving course. LiDAR and computer vision techniques were extensively used in the completion of the challenge (Behringer, R., 2005). In 2007, Carnegie Mellon University was able to modify a Chevy Tahoe to complete the DARPA grand challenge (Markoff, J., 2007). Several participants from these challenges built the Google self-driving car in 2008 (Birrsall, M., 2014). The project, now re-branded as Waymo (Waymo, 2020), performs active research in developing self-driving cars these days. Companies such as Uber (Uber, 2020) and Tesla (Tesla, 2020) are much invested in developing AVs for civilian applications. Pratt Miller Defense (Fontaine, J. 2020), Oshkosh defense (Oshkosh Defense, 2020), and Lockheed Martin (Lockheed Martin, 2020) are notable companies among others that cater to the development of AVs in the military sector.

Traffic Simulators and AV Simulators

Traffic simulators, as the name indicates, mathematically model transportation systems, including items such as freeways, intersections, arterial routes, and roundabouts. These simulators are used in road traffic analysis and simulation of urban mobility. Traffic simulators are typically classified as macroscopic, microscopic, and mesoscopic. Simulations in macroscopic models usually take place on a section-by-section basis instead of tracking individual vehicles. Traffic modeling in transportation subnetworks such as freeways and surface-street grid networks belong to this category. Microscopic models simulate and track the movement of each vehicle and incorporate vehicle-to-vehicle interactions. Mesoscopic models combine macroscopic and microscopic models. AVs can be integrated within a traffic simulator to behave just as another entity generated by the simulator. These traffic simulators offer extensive features in addition to generated vehicle traffic such as road lane lines, signals, traffic lights, pedestrians, etc. Several commercial and open-source programs provide such functionality. Although not exhaustive, Table 1 shows a representative list of surveyed and available traffic simulators along with their capabilities.

Table 1. Off-the-shelf traffic simulators and their capabilities

	SUMO	MATSim	VISSIM	AIMSUN	Paramics
Architecture	32- & 64-bit (Windows, Linux, macOS)	32- & 64-bit (Linux)	64-bit (Windows)	64-bit (Windows, Linux, macOS)	32- & 64-bit (Windows)
Cost	Open-source	Open-source	Commercial	Commercial	Commercial

Simulator model	Microscopic	Mesoscopic	Microscopic	Microscopic	Microscopic
Network communication	TCP	- NA -	COM	TCP	TCP
3rd party visualization support	Unity3D	Simunto Via, OTFVis	Unity3D	- NA -	- NA -
Programming Language	C++	Java	C++, Python, Matlab	C++	C++
Reference	(Krajzewicz, D. 2002) (Behrisch, M. 2011)	(Horni, A. 2016)	(Fellendorf, M. 2010)	(Casas, J. 2010)	(Cameron, G. D. 1996)

Only a few of the surveyed traffic simulators support the integration of an AV entity controlled by an external program. For example, the VISSIM traffic simulation software, from the PTV group, can embed up to 1000 externally connected vehicle entities such as physically rigged cars, bikes, or AVs. Driving parameters such as steering and throttling from these external entities can be sent via a Component Object Model (COM) protocol, and are simply recognized and computed as new interactable traffic entities within VISSIM.

AV simulators such as CARLA (Dosovitskiy, A. 2017), Microsoft AirSim (Shah, S., 2018), and the NVIDIA DRIVE Platform (NVIDIA DRIVE Platform, 2020)/NVIDIA DRIVE constellation (NVIDIA DRIVE Constellation, 2020), Gazebo (Gazebo, 2020) help facilitate autonomous driving research. Most of these simulators provide a platform for algorithmic development, training, and validation of autonomous urban driving systems. Additionally, they offer other capabilities that distinguish one from another. For example, the NVIDIA DRIVE Platform includes a standalone in-vehicle hardware capable of integrating vehicle sensors and processing images. Combined with software and deep neural network training, the platform is capable of generating driving controls for an AV in real-time.

Most physical self-driving cars in their development use sensors such as cameras, radars, and LiDARs to sense the surrounding world. Images and data captured from these sensors are processed using a combination of GPS information, computer vision, and machine learning techniques, in real-time, to identify objects such as vehicles, road lanes, pedestrians, etc. For example, Waymo uses a map-based approach to achieve full self-driving capabilities, whereas Tesla uses a vision-based approach (Alvarez, S., 2020). A reality of this research area is that no matter what approach is used, it is impossible to build a system that will not fail under some circumstances (Davies, A. 2019). Hence, regulatory hurdles prevent companies from incorporating full self-driving autonomy. Several commercial companies such as Acura, Audi, BMW, Cadillac, Subaru, and others offer partial self-driving capabilities such as lane assist and centering, hands-free steering, and adaptive cruise control features (Mays, K. 2019). The development and implementation of partial or full-self driving features are proprietary and primarily business driven to maintain commercial competitiveness. Hence, algorithms and data are mostly unavailable outside of companies for further research, analysis, and development. As such, academic research requires code implementation, often times, from the ground up.

In a simulated AV, virtual cameras mounted on top of the vehicle perform the role of the image capturing. These images are processed exactly like the images from a real self-driving car. When a simulated AV is placed amidst vehicle traffic generated from a traffic simulator application like VISSIM, the system functions as a physical self-driving car in the real-world. The work presented in this paper uses this approach.

METHODOLOGY

System Architecture

The system architecture for integrating AVs and traffic simulators into a multimodal traffic simulation is shown below (See figure 3). It was designed and created for this research.

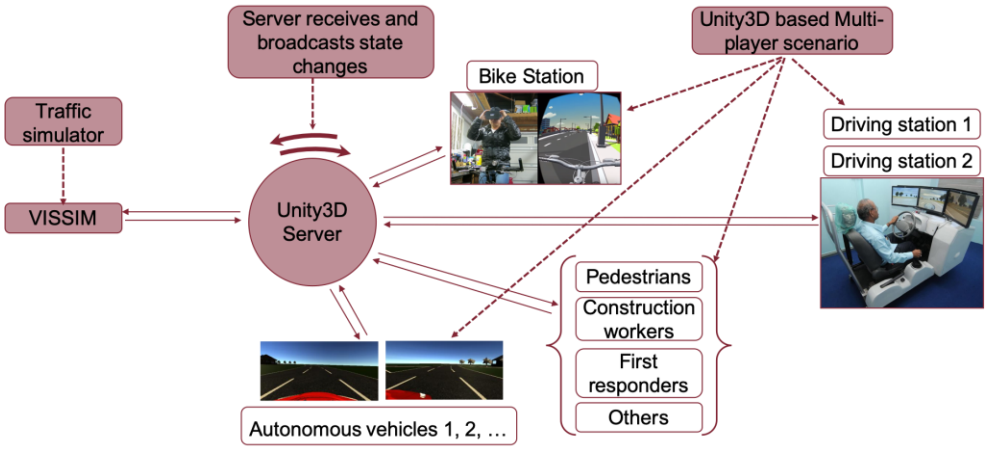


Figure 3. Multimodal Traffic Simulation Framework Architecture

As the figure portrays, the architecture has three components: a) a traffic simulator, b) a server that does bi-directional broadcasting of state changes and c) clients that represent physical vehicle rigs and AVs. The client-server implementation was made using the game engine Unity, which also doubles as the visualization platform on desktop and virtual reality environments. For a given driving environment such as an urban neighborhood or a highway, an equivalent computational road network was constructed within VISSIM. This road network was then used to generate vehicle and pedestrian traffic entities in VISSIM. The states of each vehicle entity were sent to the Unity server. The server then broadcasted these updates to each client. The clients themselves are self-aware of their position and direction, so, for example, a bike rider can automatically track the location of an AV within the scenario. The driving states (i.e., position and orientation) of the clients are sent to VISSIM via the Unity3D server every frame. These client states are processed and computed by the traffic simulator as new entities within VISSIM.

AV Design

The AV implementation was performed on a single lane road and a three-lane highway, with no other vehicle or pedestrian traffic. This was primarily intended as the development of a proof of concept machine learning model using supervised and imitation learning approaches that results in an AV staying true to the road track without showing any abnormal behavior. The Python programming language, along with its suite of data analysis and machine learning modules, was used in building the trained model. Aspects such as other vehicle traffic, pedestrians, and traffic signs were not implemented.

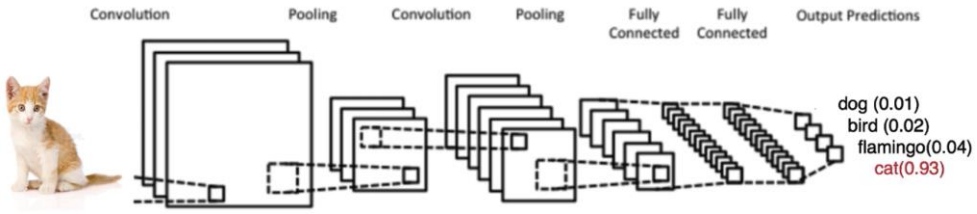


Figure 4. A Simple Convolutional Neural Network Pipeline (Figure courtesy: github.com)

Deep learning using Convolutional Neural Networks (CNN or ConvNets) was implemented in training the AV. CNNs are a special category of Artificial Neural Networks (ANN) that are highly effective at image recognition and classification (Hijazi, S., 2015). These include features such as faces, objects, traffic signs, and road lanes – the building blocks for implementing an AV. CNNs can recognize useful patterns within images by understanding relevant spatial structure that describes them as well as require fewer parameters than traditional ANNs. The primary structure of a CNN pipeline includes an input layer, convolutional layers, pooling layers, fully-connected layers (also called dense layers), and an output prediction layer. Figure 4 shows a simple CNN pipeline where the pixel values of an

input image of an animal are passed through convolution, pooling, and fully connected layers before predicting the probability of the animal type belonging to a specific class. The CNNs differ from regular neural networks due to the 'Convolution' layer within the pipeline.

The goal of the convolutional layer is to extract and learn specific image features that help classify an image. For an input image, each pixel corresponds to an input node that will be processed by a convolutional filter called the kernel (also called a kernel matrix or a filter or a feature detector). The kernel matrix consists of integers and is generally small in spatial dimensionality (e.g., 3 x 3). Convolution is performed by sequentially sliding the kernel at every location on the input image by a specific stride length. With a stride of 1, the convolution is achieved by sliding the kernel filter one pixel at a time. During convolution, every cell in the kernel matrix is multiplied element-wise by a corresponding pixel value in the input image, and a summation of values is performed (i.e., a dot product). The result is averaged and stored in a 'convolved map' or a 'feature map'. A larger stride length will result in a smaller feature map, and the feature extraction will be less accurate. Figure 5 shows an illustration of the convolution process, where an input image has a mix of gray and white pixel values. Upon applying the kernel matrix to the first set of 3x3 pixels in the input image and performing a convolution, the first cell in the feature map resulted in a -6.1. An analogy of this process can be made to feature extraction in images captured by a camera on an AV, where a value of -6.1 indicates the feature is an asphalt road, and a value of 45.5 indicates white road lines. The convolution operations make distinctive image features stored in a numerical format. In this example, the feature map was able to extract a specific feature of interest, which is to show the distinction between lower and higher pixel intensity while preserving all the image features. The feature map output will change for different kernel filter values. For example, the kernel matrix in Figure 6 is used for performing image sharpening. With different kernel matrix values, various features maps such as edge detection, gaussian blur, and box blurs can be constructed. The pooling layer in the CNN pipeline acts to shrink the image stack by reducing the dimensionality of the representation of each feature. This layer reduces the computational complexity of the model and avoids overfitting the model, while still retaining the significant characteristics from within the feature map. A series of convolutional and pooling layers further help filter the image and extract essential features. With a deeper CNN, feature maps become visually unrecognizable, but more complex patterns get encoded into the model that helps identify the input image class.

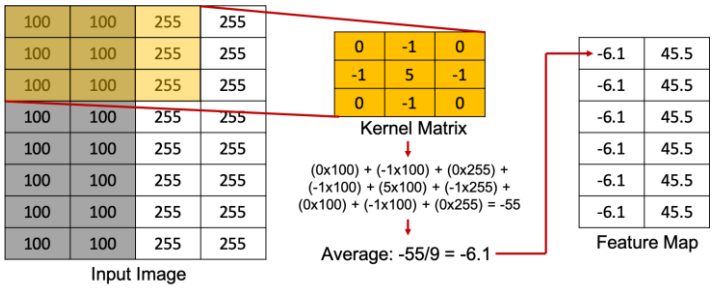


Figure 5. CNN Convolution Process

While the combination of convolution and pooling in the first part of the CNN pipeline performs feature extraction, the second part of the CNN is a fully connected layer that performs feature classification. The output from the convolution and pooling layers are flattened into a 1-D array of pixels. They serve as an input to the fully connected network, where each pixel corresponds to a node in the input layer (Figure 6). The fully connected network processes these features and computes the final probability of the class that an input image belongs to.

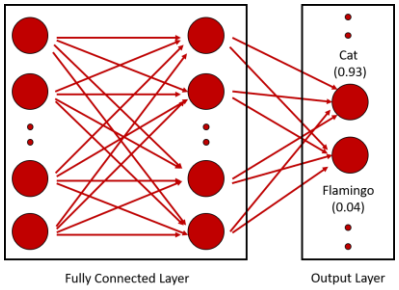


Figure 6. Feature Classification

In the work presented in the paper, two different networks were created: a) A 9-layer network with five convolutional layers and four fully-connected (dense) layers without batch normalization and b) A 14-layer network with five convolutional layers, each with corresponding batch normalization, followed by four dense layers. Batch normalization (Ioffe, S., 2015) is typically used in improving the performance and stability of a neural network. The normalization process re-centers and re-scales the input layer, i.e., the convolutional layer preceding it, in this case. Both these networks were used for training the AV on a single lane and a three-lane road track.

Data was captured as the car was manually driven three times each in a clockwise and counterclockwise direction along the outer perimeter of the road track. Driving both directions yielded non-zero steering values for both left and

right turns. A histogram of steering angles between -1 and +1, centered around a steering angle of 0, grouped into 25 bins is shown in Figure 7a. A high value for a zero-steering angle in the figure indicates that the car was predominantly driving a straight path with only occasional left or right steering. Too many data points with zero steering angle could potentially result in a trained AV to be biased in driving straight regardless of the track. However, the road track used for training and deployment were identical, making the straight path dominance in the trained model a potential benefit. Hence, two training models were developed, one with straight path dominance without any outlier removal, and another with reduced straight path dominance. The bias was reduced by removing excessive zero-steering data points while keeping a reasonable straight path dominance (see Figure 7b). Of the ~5,900 data points captured overall for training, ~3,000 were retained after straight path bias removal. In both cases, the datasets were split 80%-20% for training and validation. The distribution of steering angles for the training and validation datasets for reduced straight path dominance is shown in figure 8, which indicates that there is approximately equal representation of left and right steering angles in the data.

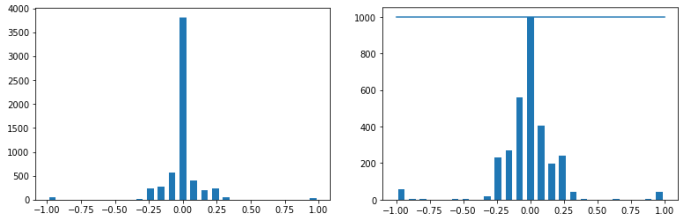


Figure 7. (a) Straight path bias in data captured (b) Straight path dominance reduced

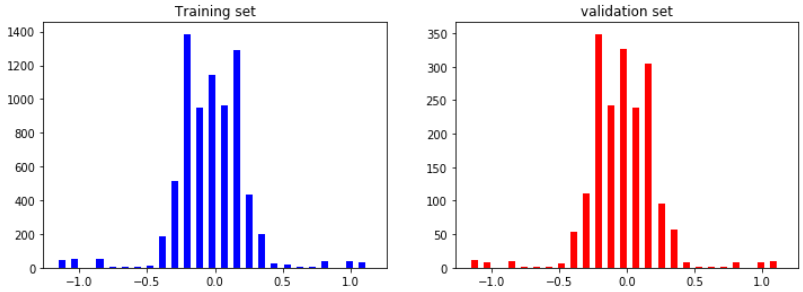
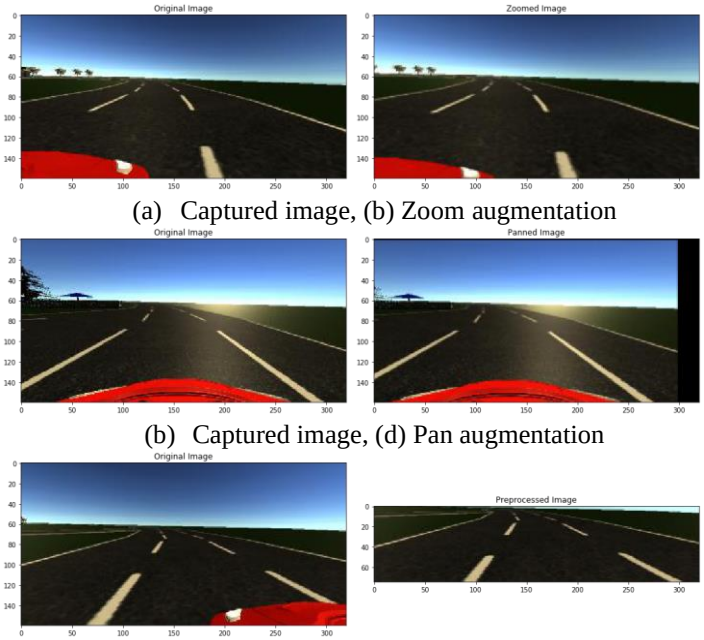


Figure 8. Histograms (a) Training dataset, (b) Validation dataset

The captured left, center and right images were pre-processed to prevent any additional data bias. Image augmentation techniques such as random zoom, pan, flip, brightness alteration were implemented at a 0.5 probability for all captured images to increase diversity in the dataset resulting in a more robust model. This prevents the likelihood of the AV drifting off-road and creates the effect of input images being in various offsets from the center lane. The purpose of these artificial offsets is to make the neural network more robust so that it can recover from a poor position or orientation more quickly and in a broader range of circumstances. The images were further auto-cropped to only keep relevant road data and remove cultural data such as the sky and the surrounding neighborhood. Figure 9 (a – f) shows a series of augmentation techniques applied to the input images. A multi-layer convolutional neural network, proposed by NVIDIA (Bojarski, M., 2016) was implemented for building the machine learning model. This method is based on the CNN pipeline described earlier and provided an architecture specifically designed for predicting AV steering values. In this method, the weights of the network were trained to minimize the mean squared error between the steering data from manual trained data/augmented images and the steering values created by the network.



(a) Captured image, (b) Zoom augmentation

(b) Captured image, (d) Pan augmentation

(e) Captured image, (f) Cropped image

Figure 9. Image Augmentation Techniques

TEST CASE SETUP

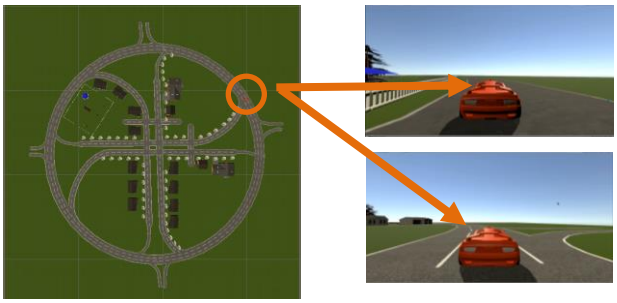


Figure 10. Aerial view of test road track and the training car on single and three lanes

Figure 10 illustrates the scenario used for testing. The training process for the AV included capturing images from three forward-facing virtual cameras positioned left, center, and right on top of a car. As the car was manually driven along the desired path within the scenario, images from the cameras and various driving parameters (steering, throttle, reverse, and speed) were captured every frame. The goal was to predict the steering value (independent variable), given a single center camera capture mounted on the virtual AV (dependent variable). Clockwise or counterclockwise, the track has four exits leading to the town's local neighborhood and four exits leading outside of the region. Also, given the size of the track compared to the AV, much of the test track is a straight path with only occasional left- or right-turns. As such, it became necessary to investigate the influence of zero-steering straight path bias in the trained AV, as discussed earlier.

On the other hand, batch normalization, as discussed in the methodology section, adds computational time to build a machine learning model and hence make steering predictions. However, it is unclear how much value it will add for the test case set up. To investigate the influence of batch normalization, two sets of test cases were developed, one with and another without batch normalization. For each AV deployed on single- and three-lane roads, the following metrics were evaluated: i) model training time, ii) lane deviations, iii) incorrect exiting onto a local road, iv) accurate path correction after deviation and v) staying on the road. A wall timer that measures elapsed time during the machine learning model build was implemented to evaluate model training time. The metrics (ii) – (v) were visually assessed by a human for each test case three times as the AV auto navigated around the road track.

The AV on the single lane road was trained on a PC with an Intel i7 3.6 GHz processor, 32GB memory, and an NVIDIA GeForce GTX 2060 GPU. The AV on the three-lane road was trained on a PC with an Intel i7 4.2 GHz processor, 16GB memory, and an NVIDIA GeForce GTX 1070 GPU. The Python module Keras, an easily accessible deep learning API built using Tensorflow, was used to create the AV machine learning models. Images cropped, as seen in Figure 10f, at the end of the image augmentation technique were converted from RGB to YUV color space for data processing. YUV colorspace allows for reduced bandwidth for chrominance components in the images and is more efficient than RGB representation. Since the augmented dataset was reasonably large and potentially exhausted system memory resources while creating a machine learning model, a fit generator was used in batches and not on the entire dataset. Error minimization was performed on the training dataset over ten epochs using Python Keras' Adam optimizer (Kingma, D.P., 2014). Adam is a first-order gradient-based method used in the optimization of stochastic objective functions. It is computationally efficient, has fewer memory requirements, and is a popular optimizer used in the machine learning community.

The AV captured about 6,000 images during the training process. The batch generation strategy used created subsets of 100 images each, from the training dataset, with a limit of 10 epochs (generations), and 300 steps (iterations) per epoch. Trained AV models were tested using a drive controller, implemented in Python. The drive controller loaded the machine learning model and calculated the steering angle for images received from the AV via socket communications. The images themselves were captured from a virtual camera mounted top center of the AV facing forward.

RESULTS AND DISCUSSION

The operating characteristics of the AV in both the single-lane road and three-lane highway were independent of each other and not correlated. Hence, the evaluation metrics were conducted for each scenario independently and tabulated (Tables 2 and 3). As stated earlier, the defined metrics (ii) – (v) were visually evaluated by a human for each test case three times as the AV auto navigated around the road track.

Table 2. Test scenario setup for single-lane AV (3.6 GHz CPU, NVIDIA GeForce GTX 2060)

Cases / Metrics	With CNN batch normalization		Without CNN batch normalization	
	With zero-steering bias	**Reduced zero-steering bias**	**With zero-steering bias**	**Reduced zero-steering bias**
Model training time	22.1 minutes	22.3 minutes	20.6 minutes	20.5 minutes
Lane departures?	No	No	No	No
Did the AV exit into a local road?	No	No	No	No
Did the AV perform path correction?	Did not stray	Yes, recovered from the green grass area	Did not stray	Did not stray
Did the AV wander off?	No	No	No	No

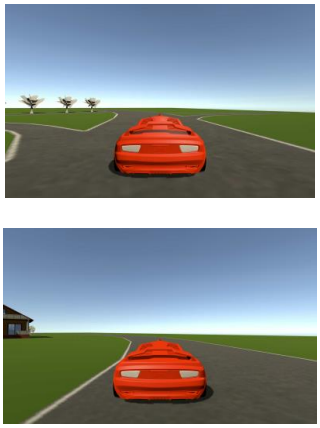


Figure 11. AV on Single Lane: (a) Auto steering at the center (b) Leaning towards the left

It can be seen from Table 2 that the training time for the AV model built using Convolutional Neural Network (CNN) with batch normalization took longer than without batch normalization. This is expected because the batch normalization process computes the mean and variance of each mini-batch of data and normalizes each feature based on the mini-batch statistics, adding more computational time. In each of the investigated cases within this scenario, the AV consistently stayed on and successfully navigated the single-lane road. In one instance with reduced zero-steering bias, however (i.e., reduced straight path dominance), the AV strayed from the single lane into the grass area once, but instantly recovered from it and continued driving on the single-lane road. Figure 11 shows screenshots of the AV auto-steer on the single lane track. For this scenario, the AV did show an inclination to stay on the left side of the road, indicating that the images with exits to the local neighborhood might have factored in when determining the steering angle during experimental runs.

Table 3 below shows the AV evaluation metrics on the three-lane highway scenario. The mesh geometry for both the single-lane and three-lane scenarios were identical. The only variation in the three-lane scenario was the road texture with multiple lane markings, making it a distinctive scenario for the AV training purpose. Four rounds of training were performed with/without CNN batch normalization, and with/reduced zero-steering bias.

Table 3. Test scenario setup for three-lane AV (4.2 GHz CPU, NVIDIA GeForce GTX 1070)

Cases / Metrics	With CNN batch normalization		Without CNN batch normalization	
	With zero-steering bias	**Reduced zero-steering bias**	**With zero-steering bias**	**Reduced zero-steering bias**
Model training time	48.5 minutes	47.9 minutes	45.7 minutes	45.8 minutes
Lane departures?	Continuous fishtailing but stayed in the lane	Fishtailed, and deviated	None	None
Did the AV exit into a local road?	No	Yes	No	No
Did the AV perform path correction?	Corrected instantly	Eventually drove back to the 3-lane road	Instant correction when the edge of lane marking reached	No correction required
Did the AV wander off?	No	No	No	No

Due to the use of a lower-powered GPU, the model training on the three-lane highway was slower than the single-lane road. Training without batch normalization, especially with reduced zero-steering bias (i.e., reduced straight path dominance), resulted in very smooth AV steering within the middle lane (i.e., no sudden path corrections). The AV was trained while in the middle lane of the three-lane highway. On the case with zero-steering bias (i.e., the entire image dataset with all zero-steer angles), the AV stayed in the middle lane but showed an inclination to drive straight until the curve of the road was realized, before performing an instant correction. Figure 12 shows the screenshot of the AV auto-steering without CNN batch normalization, and with reduced zero-steering bias.

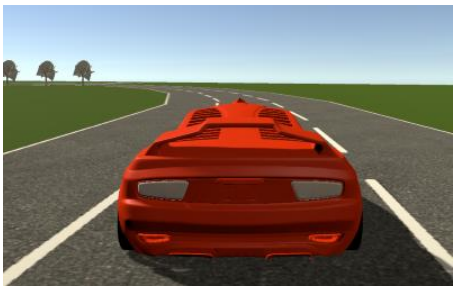
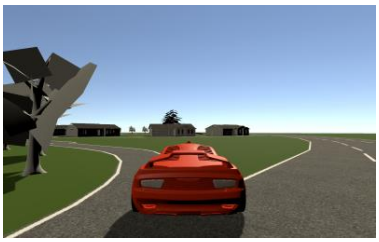


Figure 12. AV trained without batch normalization, reduced zero-steering bias

(a) With zero-steering bias



(b) Reduced zero-steering bias, before taking exit

(c) Reduced zero-steering bias, after taking exit

Figure 13. AV with batch normalization

The results were interesting when batch normalization was implemented within the training process (Figure 13). With the zero-steering bias case, the AV continuously fishtailed but stayed true to the middle lane that it was trained on, with instant path correction (Figure 13 a). With reduced zero-steering bias (i.e., reduced straight path dominance), the AV deviated from the middle lane and took the first left exit it encountered and drove into the local neighborhood. During this stage, the AV stayed true to the asphalt road and never veered off into the grassy area or the houses (Figure 13 b and c). This behavior, exiting the highway, was unexpected, since batch normalization has traditionally been attributed to improved speed, performance, and stability of artificial neural networks. In an ideal network, the global mean and variance would be a preferred feature to normalize the inputs to a layer. However, it is computationally expensive to arrive at these values after each update in the network. Hence, the mean and variance are estimated using mini-batch statistics. When using reduced zero-steering bias (i.e., reduced straight path dominance), the statistics for each mini-batch significantly differed from the others due to the loss of some straight path data. It was theorized that this caused the AV to veer and proceed toward the first exit it found. The AV departure from the middle lane with zero-steering bias was much less pronounced because the mean and the variance for each mini-batch was influenced by a larger set of straight path zero-steer values than the one with reduced zero-steering bias. This is an outcome that certainly requires further investigation and future work.

Given the four cases on the three-lane highway scenario, the model without CNN batch normalization and reduced zero-steering bias showed the most stability and accuracy. This AV model was hence considered for implementation within the overall traffic simulation framework.

AV Within a Traffic Simulator Framework

The traffic simulator VISSIM used in this research is capable of probabilistically generating thousands of vehicles within a road network. Each vehicle VISSIM created is capable of auto navigating based on the rules stipulated by its computational environment. VISSIM supports the integration of external vehicles within its traffic network if their positions and orientations are communicated in a specific format using a COM server architecture. The program can also transmit the position and orientation of every vehicle it generates via the same COM server protocol to third-party applications. These vehicles are not AVs as VISSIM is aware of what every vehicle is doing across the entire simulation and can provide position and orientation updates to all to avoid collisions or other unintended possibilities.

Figure 14. Integration of AV within VISSIM traffic simulator

The position and orientation information of all VISSIM entities was captured in real-time and used within a Unity visualization application. The developed AV model was integrated into the Unity client, whose data was network communicated to VISSIM using the system architecture described in the Methodology section (Figure 3). Figure 14 shows the integration of VISSIM traffic relayed to the Unity AV client through the Unity server. Both the clients and server were created in this research. The AV, identified by an oval around it, was made to auto-navigate on the three-lane highway, while VISSIM generated the vehicles seen in the vicinity.

It is to be noted that the VISSIM generated traffic was aware of the AV in its road network, except that its position and orientation was controlled by a machine learning model loaded by a Python drive controller externally. However, the AV was unaware of any traffic since it was trained only on an empty road. Therefore, as expected, the vehicles

(a) VISSIM vehicles veer around the AV

(b) AV runs into a VISSIM vehicle

Figure 15. Artifacts of AV aware VISSIM traffic but not vice-versa

generated by VISSIM were able to veer around the AV (Figure 15a). However, the AV itself ran through VISSIM vehicles that stopped at a traffic intersection (Figure 15b). Figure 15a is a representative image showing that none of the VISSIM vehicles ran into the AV since the program was aware of its position and orientation. However, as seen in Figure 15b, the AV ran into a stopped VISSIM vehicle because it was not trained to veer around an existing vehicle. The behavior of the AV in such situations was undefined. It produced effects during ad-hoc testing, such as the AV deviating from the road path and veering into the green grass area before correcting its path back to an empty road section. Ideally, computer vision techniques should be used in conjunction with a machine learning model to make the AV aware of its neighborhood that includes other vehicles, pedestrians, and traffic lights. While these situations were not accurate, they were expected. However, they do not diminish the accomplishment of a trained AV in a large, multimodal simulation.

CONCLUSIONS AND FUTURE WORK

The scenarios presented in the paper showed the development and deployment of an AV in a simulated urban neighborhood. The results demonstrated that the AV is capable of auto navigating on a track trained for deployment using simple virtual RGB cameras – three for training, and one for deployment. This means that a limited set of hardware was sufficient, in conjunction with an advanced deep learning model, to correctly steer an AV on a one and three lane simulated roadway. Although the AV was implemented on an empty road (i.e., no other traffic or pedestrians), the work presented in the paper is a proof of concept pre-work for a fully developed machine learning model to recognize other vehicles, road signs, and pedestrians within a realistic traffic simulation framework. This is one of the next tasks to be completed in future work. Another area for future work is to quantifiably investigate the influence of CNN batch normalization in determining the steering angle.

This work can readily be extrapolated to military applications, where a battlefield scenario can replace the urban neighborhood scenario. An army vehicle such as Humvee can be trained to auto-steer in the scenario with or without CNN batch normalization. When the battlefield road path is predominantly straight with no unexpected curved paths, it is recommended that reduced zero-steer bias be implemented for training. On the other hand, if the battlefield road path is curvy, it is recommended that the training be implemented with zero-steer bias.

ACKNOWLEDGMENTS

The authors would like to thank the Turner-Fairbank Highway Research Center for providing funding support on the project. Project support from the Institution of Transportation at Iowa State University is appreciated.

REFERENCES

Alvarez, S. (2020). *Tesla's controversial vision-based full self-driving approach is finally paying off*. Retrieved June 21, 2020. Retrieved from https://www.teslarati.com/tesla-autopilot-full-self-driving-vision-based-approach-validated-video/

Behringer, R., Sundareswaran, S., Gregory, B., Elsley, R., Addison, B., Guthmiller, W., Daily, R., Bevly, D., (2004). *The DARPA Grand Challenge – Development of an Autonomous Vehicle*, IEEE Intelligent Vehicles Symposium, Parma, Italy, pp. 226-231, DOI: 10.1109/IVS.2004.1336386.

Behringer, R., Travis, W., Daily, R., Bevly, D., Kubinger, W., Herzner, W., & Fehlberg, V. (2005, September). *RASCAL -an autonomous ground vehicle for desert driving in the darpa grand challenge 2005*. In Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005. (pp. 644-649). IEEE.

Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D. (2011). *SUMO-simulation of urban mobility: an overview.* In proceedings of SIMUL 2011. The Third International Conference on Advances in System Simulation. ThinkMind.

Birdsall, M., (2014). *Google and ITE: The Road Ahead for Self-Driving Cars.* Institute of Transportation Engineers, ITE Journal, May 2014; 84, 5: ABI/INFORM Global

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J. and Zhang, X. (2016). *End to end learning for self-driving cars*. arXiv preprint arXiv:1604.07316.

Cameron, G. D., Duncan G. I. (1996). *PARAMICS – Parallel microscopic simulation of road traffic.* The journal of supercomputing, 10(1), pp. 25-53.

Casas, J., Ferrer, J. L., Garcia, D., Perarnau, J., Torday, A (2010). *Traffic simulation with aimsun.* In fundamentals of traffic simulation (pp. 173-232). Springer, New York, NY.

Davies, A. (2019). *Tesla's latest auto-pilot death looks just like a prior crash.* Retrieved June 21, 2020 from https://www.wired.com/story/teslas-latest-autopilot-death-looks-like-prior-crash/

Demaitre, E., (2019). *Driverless investment tops $1.6B so far this month*. Retrieved May 22, 2020, from https://www.therobotreport.com/driverless-investment-high-valentines/

Dhawan, C. (2019). *Autonomous Vehicles Plus: A Critical Analysis of Challenges Delaying AV Nirvana.* FriesenPress, ISBN: 9781525539848.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V. (2017). *CARLA: An open urban driving simulator*. arXiv preprint arXiv: 1711.03938.

Fellendorf, M., Vortisch, P. (2010). *Mircoscopic traffic simulator VISSIM.* In fundamentals of traffic simulation (pp. 63-93). Springer, New York, NY.

Fontaine, J. (2020). *Pratt Miller Defense Provides Robotic Platform for Army Evaluation.* Retrieved May 24, 2020, from https://www.prattmiller.com/news/article/524

Gazebo (2020). *Gazebo Vehicle and City Simulation*. Retrieved Jun 22, 2020 from http://gazebosim.org/blog/car_sim

Green, H., (1925). *Radio Controlled Automobile.* Electronic World Magazine. Retrieved June 17, 2020, from http://www.americanradiohistory.com/Archive-Radio-News/20s/Radio-News-1925-11-R.pdf

Hicks, N., (2017). *Nebraska Tested Driverless Car Technology 60 Years Ago.* Lincoln Journal Star, Sep 13, 2017, Page A1. Retrieved June 17, 2020, from https://journalstar.com/news/local/govt-and-politics/nebraska-tested-driverless-car-technology-60-years-ago/article_a702fab9-cac3-5a6e-a95c-9b597fdab078.html

Hijazi, S., Kumar, R., & Rowen, C. (2015). *Using convolutional neural networks for image recognition.* Cadence Design Systems Inc.: San Jose, CA, USA, 1-12.

Horni, A., Nagel, K., Axhausen, K. W. (2016). *The multi-agent transport simulation MATSim.* London: Ubiquity Press.

Ioffe, S., & Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* arXiv preprint arXiv:1502.03167.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

Krajzewicz, D., Hertkorn, G., Rössel, C., Wagner, P. (2002). *SUMO (Simulation of Urban Mobility) – an open-source traffic simulation.* In proceedings of the 4th middle east symposium on simulation and modeling (MESM20002), pp. 183-187.

Leighty, R. D. (1986). *DARPA ALV (autonomous land vehicle) summary.* NO. ETL-R-085. ARMY ENGINEER TOPOGRAPHIC LABS FORT BELVOIR VA.

Lockheed Martin. *Driving Forces: Autonomous Land Vehicles.* Retrieved May 23, 2020, from https://www.lockheedmartin.com/en-us/news/features/history/alv.html

Lockheed Martin Defense (2020). *Autonomous and Unmanned Systems | Lockheed Martin.* Retrieved June 17, 2020, from https://www.lockheedmartin.com/en-us/capabilities/autonomous-unmanned-systems.html

Markoff, J., (2007). *Crashes and Traffic Jams in Military Test of Robotic Vehicles.* The New York Times, Nov 5, 2007. Retrieved June 17, 2020, from https://www.nytimes.com/2007/11/05/technology/05robot.html

Mays, K. (2019). *Which Cars have self-driving features for 2019?* Retrieved June 21, 2020 from https://www.cars.com/articles/which-cars-have-self-driving-features-for-2019-402645/

Muller, J., (2019). *The Army Steps up its Pace on Self-Driving Cars*. Retrieved May 22, 2020, from https://www.axios.com/us-army-military-casualties-autonomous-vehicles-1ff51e01-3b16-4a1c-9587-ce55dee74788.html

NVIDIA DRIVE CONSTELLATION. (2020). NVIDIA DRIVE CONSTELLATION – Virtual Reality Autonomous Vehicle Simulator. Retrieved June 17, 2020 from https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

NVIDIA DRIVE Platform. (2020). NVIDIA DRIVE – Autonomous Vehicle Development Platforms. Retrieved June 17, 2020 from https://developer.nvidia.com/drive

Oshkosh Defense (2020). *Terramax Unmanned Ground Vehicle Technology | Oshkosh Defense.* Retrieved June 17, 2020, from https://oshkoshdefense.com/advanced-technologies/terramax-unmanned-ground-vehicle-technology/

Paleofuture – The history of the future (2007). *DARPA Spent $1 Billion Trying to Build a Real-Life Skynet in the 1980s.* Retrieved May 23, 2020, from https://paleofuture.com/blog/2019/4/30/darpa-spent-1-billion-trying-to-build-a-real-life-skynet-in-the-1980s

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2018). *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*. In Field and service robotics (pp. 621-635). Springer, Cham.

Tesla (2020). *Tesla Home page.* Retrieved June 17, 2020, from https://tesla.com/

Uber (2016). *Uber's first self-driving fleet arrives in Pittsburgh this month. Bloomberg.* Retrieved June 17, 2020, from http://www.bloomberg.com/news/features/2016-08-18/uber-s-first-self-driving-fleet-arrives-in-pittsburgh-this-month-is06r7on

(United States Senate committee on armed services transcript) (2018). *Hearing to Receive Testimony on Accelerating New Technologies to Meet Emerging Threats*. Docket 18-40_04-18-18.

Waymo, (2020). Waymo home page. Retrieved June 17, 2020, from https://waymo.com/

Waxman, A., LeMoigne, J., Davis, L., Srinivasan, B., Kushner, T., Liang, E., Siddalingaiah, T. (1987). *A Visual Navigation System for Autonomous Land Vehicles.* IEEE Journal on Robotics and Automation, vol. 3, No. 2., pp 124-141.