Shallow Neural Network Control for Off Road Autonomous Vehicles via Simulation

Matthew Paul Spencer
Keweenaw Research Center
Michigan Technological University
Houghton, MI 49931-1295, United States of America
mpspence@mtu.edu

***Abstract*—Simulation has been an ongoing area of interest regarding predictive path control planning for autonomous off road vehicle platforms. Using a simulated world for predictive path planning, an autonomous vehicle platform can perform analysis of multiple possible routes across outdoor terrains regardless of weather and soil conditions due to its programmable static nature. By utilizing real world terrain telemetry alongside an overlay of accurate aerial imagery, this work evaluates utilizing a trained shallow feed-forward neural network to control a simulated Unreal Engine 4 model of an autonomous vehicle moving across a simulated virtual terrain.**

***Index Terms*—autonomous, vehicle, off-road, UE4, simulation, neural, network**

I. Introduction

A. Problem Statement

Simulation research regarding Neural Networks is relatively new in the area of autonomous vehicles (AV), and an emerging field of study in controlling off road AV platforms. The unstructured and hazardous terrains that off road AVs must traverse through makes the problem of creating simulated path planning or control solutions applicable to real-world AVs more difficult than its structured on-road platform counterpart. Many simulated solutions that exist today utilize highly complex federated systems that must take into account long processing times to evaluate different physics systems and components of an AV model in parallel with implementing algorithmic systems for AV control. Neural networks have recently seen use in conjunction with AV simulations by training or testing real-life AV systems within virtually created landscapes, keeping the control and path planning fields of research separate.

A complete real-time solution for AV control using a simulated model would be a large benefit to current AV research, as simulated worlds could be used for the primary path planning and traversal functionality instead of needing the time and money to implement the experiments within the real-world. Other benefits of using a simulated model to control and path plan for AVs is that simulations are unaffected by outside variables such as weather, leaving the immediate surrounding threats caused by unknown variables in potential real-world hybrid applications to be handled by an AV's on-board sensory system.

This paper analyzes an off road AV neural network control solution utilizing a virtual simulation built with modern-day computer graphics and simulated physics technology. It is theorized that by using a simplified shallow neural network trained on the virtual terrain's aerial imagery, an AV can successfully traverse through an off road virtual environment in real-time.

B. Overview

The wide adoption of AV platforms remains in their infancy, as vehicles today are implementing autonomous features that are meant to assist human drivers using Radar, LiDAR, and camera technologies rather than completely take control of the vehicle. These assisting features remain an area of immediate research [1]. Many on-road solutions from reputable technology giants such as Google and Tesla are creating proprietary neural network and artificial intelligence technology to leverage image-recognition software within AV platform systems [2]. Until this technology is perfected to a level of certainty to ensure customers of their safety, along with the legal regulation process that is currently underway at the time of writing, AV technology still remains a ripe area of research and investment for both large and small corporations and government bodies across the world.

As AI technology comes into its own, 3D computer simulation tools and game engines such as ANVEL, Car-Sim, Unreal Engine, Gazebo, Unity, etc. [3] have been used to carry out AV simulation experiments in regards to path planning and control algorithms for the past few decades. Many of the simulations created have been complex, requiring many computers and hours of executing simulation cycles to complete one simulation [4].

Couple these past simulation approaches with the time it takes for many deep neural network architectures to learn new outdoor environments, the development of any reliable real-time application has been only a possibility with past technology. But as computer simulation software becomes better optimized and malleable due to the open source nature of many graphics and physics libraries, such as with the Unreal Engine, implementing a pretrained neural network (NN) to assist in controlling an AV platform within a simulated virtual environment is very achievable today in a reasonable timeframe.

This paper creates a virtual representation of a portion of real-world terrain telemetry acquired from the outdoor

test-course at Michigan Technological University's Keweenaw Research Center (KRC) within Unreal Engine 4 (UE4). This terrain is used to train and test a shallow NN, whose purpose is to recognize desirable terrain features that a modeled AV platform can traverse over during operation.

The shallow NN used in this paper is created using MATLAB's "Deep Learning Toolbox" [5], and trained over the selected terrain aerial image KRC data. The image data is modified by a human user to indicate which terrain is traversable or untraversable, and preprocessed into gray-scale for quicker training of the NN. The trained NN is then converted into usable C++ code to be used within the UE4 AV controller.

In the next section, previous work regarding off road AV simulations and NNs used within virtual simulations are discussed, followed by the neural network setup and the simulation's overall implementation. The paper then concludes whether the trained shallow NN was successfully able to guide the vehicle over traversable terrain within the virtual environment, where data is gathered regarding what percentage of the terrain during the AV platform's execution was traversable terrain.

II. Background

A. Previous NN Simulation Work

Most work that has been done recently utilizing NNs and computer simulations was in regards to either training a NN architecture or testing a pre-trained NN within a virtual simulated environment.

For example, a plugin has been made for UE4 called "UnrealGT" which exports ground truth data sets created within UE4 using the native landscape and mesh editing tools. These ground truth data sets are then used to train NNs, which therefore are trained on user generated terrain and are not representative of real-world imagery or terrain data. It was found that the NN models trained on these data sets achieved poor results when tested on real-world data sets, which was to be expected [6].

D. Talwar et al. also found that the translation between training or testing a NN within a virtual world and trying to apply the NN to a real-world data set was very challenging. They determined that using simulated data at any step within the NN training or testing process is a "questionable" endeavour. [7].

These approaches mentioned above attempt to use human generated virtual environments within their simulations, and do not marry the real-world and virtual-world. This paper implements a geo-located point cloud with terrain aerial imagery of the KRC test-course in hopes that the NN can achieve better success training within the virtual world which accurately represents the real-world environment it is modeled after.

B. Unreal Engine 4

UE4 has been used to great extent recently as a visual simulation tool within the AV research community. Its popular use among video game developers provides a highly populated online community which regularly gives feedback and documentation alongside steady UE4 engine updates [8]. The source code is also made publicly available by the developers, allowing programmers with a good understanding of C++ and the necessary skills to develop engine specific tools and plugins to achieve specific simulation related goals [4]. The underlying physics framework PhysX is also well known and documented, providing a malleable foundation only bound by development time and the human resources required for development.

A recent UE4 build has been shown executing a default UE4 mid-sized template vehicle model to autonomously drive within a virtual environment using the proprietary MUONS path planning stack, and it was determined that UE4 handled this process well when compared to a simpler HUSKY robot platform [9]. MUONS has also been shown to be able to virtually incorporate the robot operating system (ROS) libraries to publish and subscribe to virtual sensors required to path plan adequately within complex unstructured environments. The environment created for the latter MUONS experiment was generated using a portion of real-world point cloud data taken of the KRC's outdoor test course [10].

For these UE4 experiments, all data used for path planning control was done virtually inside of the simulation without the use of any NNs. This paper seeks to incorporate a NN component to control the vehicle, and also build upon the work done in UE4 as a viable simulation tool.

III. Simulation Setup

This section explains the overall configuration of the UE4 simulation, going over the NN data collection, MATLAB setup, NN training, C++ NN exportation, and the KRC terrain importation process within UE4, and finally the control method used on the modeled AV platform for testing.

A. Specifications

- Intel i9-7920X 2.9 GHz
- 128 GB DDR4 RAM
- NVIDIA GeForce RTX 2070
- Windows 10 64-bit
- Unreal Engine 4.26.2
- MATLAB R2021a

B. Data Collection and Processing

Geo-located aerial image data from a selected portion of the KRC terrain data was processed and used as the input data to train and test the shallow NN in MATLAB. The terrain image and its corresponding heightmap file is imported as a dedicated 3D landscape tile into UE4, which is explained within a later subsection of this paper. The original 2048x2048 colored image of the terrain can be viewed below in Fig. 1.

The image is then hand edited, wherein the traversable terrain is painted over white, leaving the rest of the image black to feed as truth data for the NN shown in Fig. 2.

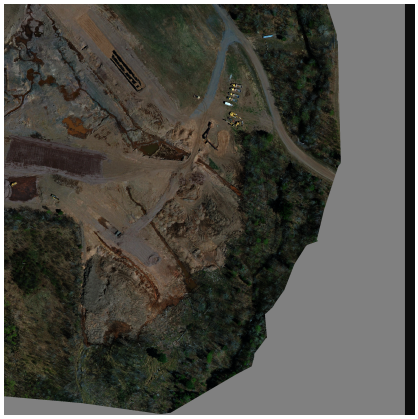

Fig. 1. Colored KRC Terrain Image

Fig. 2. Truth Image

The colored input image seen in Fig. 1 is then processed in MATLAB and converted from RGB to gray-scale, this is done to lessen the computational cost when training the NN. The gray-scale image can be seen in Fig. 3.

Fig. 3. Gray-scale KRC Terrain Image

A 3x3 averaging filter was then applied to the gray-scale image in Fig. 3. Each pixel value was then used to find the average, *avg*, and maximum, *max*, filtered pixel value, x. These values were used as threshold boundaries to create a new truth image for the NN as stated in Equation 1 and seen in Fig. 4. This was done to better identify drivable areas of the terrain based on the human chosen truth data. The gray-scale terrain image data was then used to train the shallow NN created in the process outlined in following subsection.

$$f(x) = \begin{cases} 255, & \text{if } avg \leq x \leq max \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Fig. 4. Modified Truth Image

C. Shallow NN Creation

A shallow feed-forward NN with backward propagation was created within MATLAB utilizing the "Deep Learning Toolbox" [5] as previously mentioned. The Levenberg-Marquardt, "trainlm", algorithm was selected for the learning process as it was shown to train faster than standard gradient decent for this problem set. The NN was configured to have 2 hidden layers with each layer having a width of 3. The overall MATLAB NN setup can be seen in Table 1 below.

TABLE I
MATLAB NEURAL NETWORK SETTINGS

Parameter	***Setting***
Hidden Layers	2 layers
Layer Width	3 nodes
Learning Algo.	trainlm
Epochs	100
Max Fail	1000
Min Gradient	1e-500

Each epoch consisted of training over every pixel within the gray-scale image [Fig. 3] and applying the 3x3 averaging filter to each pixel. The output is then compared against the same pixel location within the updated truth data [Fig. 4]. Each traversable pixel of terrain with a value of 255 and highlighted white was given a value of 1, and each non-drivable terrain pixel value was given a value -1.

The NN was trained for the full 100 epochs and tested on the training data. This process over-fitted the NN model (which was desired). The over-fitting was done to ensure that the vehicle would successfully detect if the upcoming terrain was traversable, in hopes that it would better prevent the vehicle from driving into hazardous areas. The final trained MATLAB model can be viewed in Fig. 5.

To validate that the NN is over-fitted, the average mean-squared error of each epoch was calculated wherein the error of the NN output to the true value of -1 or 1 for each pixel was used [Fig. 6].

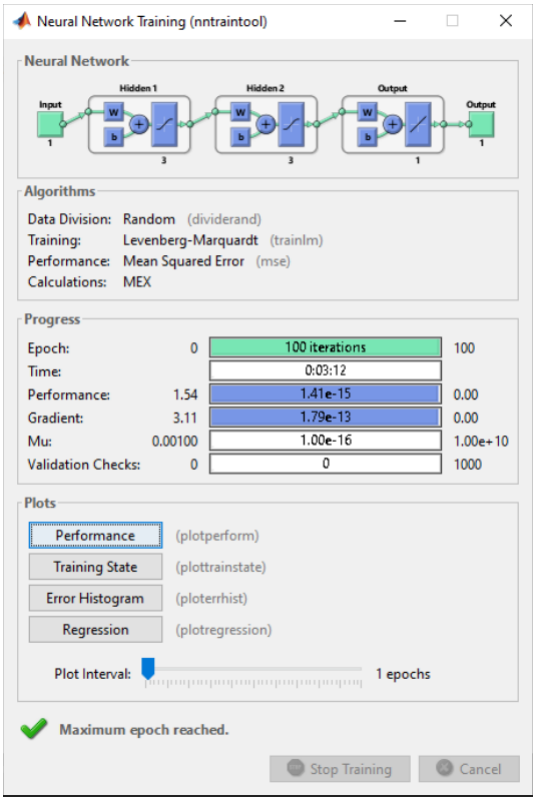

Fig. 5. MATLAB Trained Neural Network

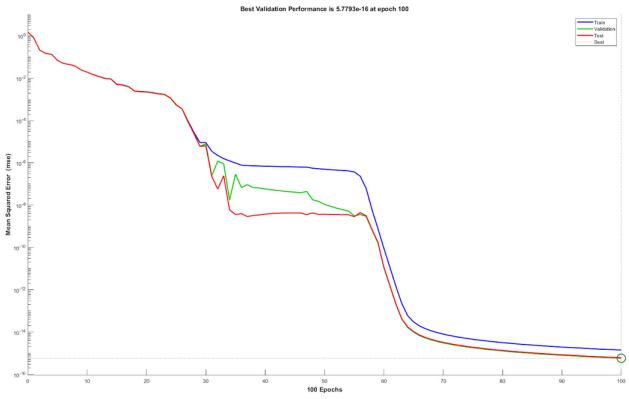

Fig. 6. Trained NN Average MSE

The trained NN was then exported and transformed into a C++ function using MATLAB's code generation feature, making the Network usable within the C++ UE4 game engine code. The function created takes a pixel value between 0 and 255, and then outputs a 1 or -1, indicating whether or not the pixel is able to be traversed over by the modeled AV platform.

D. Terrain Importation

A geo-located point cloud LiDAR data set of the KRC's outdoor test course is transformed into a static mesh within the mesh editing software Blender [11]. Geo-located drone images of the test course is then placed on top of the mesh, colorizing it. Each point in the point cloud is zeroed using its minimum (x,y) coordinate, ensuring that the terrain is compatible with UE4's world coordinate system, orienting the minimum coordinate of the terrain at the simulated world's origin.

A Python script utilizing Blender's API is then executed on the mesh, transforming and scaling the terrain mesh into 2048x2048 tiles. Each tile produced has a corresponding texture image and heightmap file, which is loaded into UE4 using a custom proprietary C++ plug-in tool "KRC Terrain Creator" [Fig. 7]. The plug-in was built using UE4 editor's API landscape creation functionality and internal engine functions. Each tile is re-scaled and textured within UE4 using the plug-in to match UE4's coordinate and unit systems. Each tile that is not the target terrain image chosen for the NN is hidden within the UE4 world, leaving only the target terrain visible within the simulation during execution.

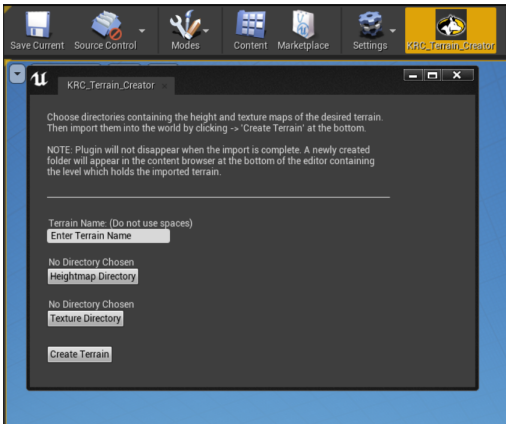

Fig. 7. KRC Terrain Creator UE4 Plugin

The generated UE4 landscape tile used for experimentation can be seen in Fig. 8.

E. Vehicle Model

A scaled model of an MRZR vehicle platform was imported into UE4 [Fig. 9] and configured based on the default advanced vehicle template supplied by the engine. A few of the UE4 vehicle parameters were tuned to correlate with the real MRZR properties as outlined in Table 2.

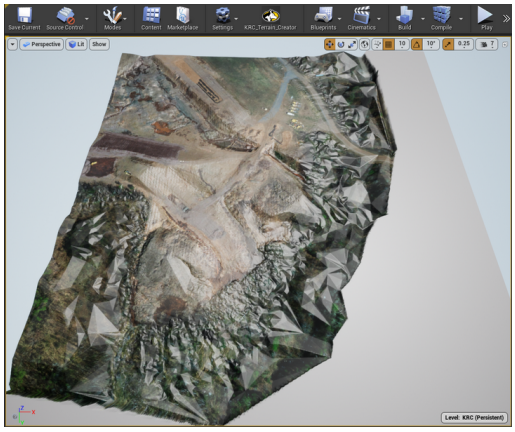

Fig. 8. KRC UE4 Landscape Tile

TABLE II
HMMWV UE4 MODEL PARAMETERS

Parameter	*Setting*
Vehicle length	360.68 cm
Vehicle width	148.59 cm

IV. EXECUTION

This section explains how the MRZR vehicle uses the NN to find a waypoint destination and move throughout the terrain during the simulation. Each simulation run is executed for 90 seconds, recording the location of where the vehicle is, and recording if it is traversing over drivable or undrivable terrain. This data is used to analyze whether the NN was successful in guiding the AV platform through drivable terrain deemed traversable.

A. Waypoint Finder

The MRZR starting location is first manually chosen before the simulation on a location within the terrain tile. During the simulation, a waypoint vector with a radius, r, of 1 meter pulses 1 degree at a time for 180 degrees in front of the vehicle. This vector moves in a half-circle motion starting at the MRZR's -90 degree point displayed in Fig 10, ending at the vehicle's 90 degree point.

Fig. 9. MRZR UE4 Model

Each world coordinate point along the curvature of the waypoint vector is captured and used to find the corresponding pixel within the gray-scale image in Fig 3. The 3x3 average filter is applied to get the new pixel value x, which is used as the input within the imported trained NN from MATLAB. The NN then returns either a 1 or -1, indicating whether the point at the specified radius and degree is able to be driven to.

If the NN returns a -1, the vector is moved along the curvature by 1 degree and the process repeats until a traversable point is found or a full 180 degrees is completed. If a destination point is found, then the steering angle θ is calculated at each simulation step to turn the vehicle in the desired direction while a hand-tuned PID controller handles the acceleration of the vehicle until the destination is reached. Once the destination is reached, the vehicle stops and repeats the process. Pseudo-code for this NN algorithm is described in Algorithm 1.

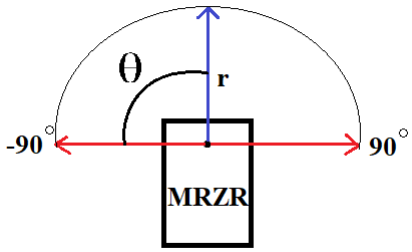

Fig. 10. MRZR Waypoint Vector

The implementation described in Algorithm 1 pads each upcoming waypoint destination, giving the vehicle a threshold of forgiveness to prevent it from missing the pixel representing the destination waypoint coordinate within the imported terrain. The area chosen for the padding is set and determined before the experiment, which was 25 cm. Thus, a target padding threshold of (25cm x 25cm) was configured for each waypoint during the simulation.

V. RESULTS

The simulation was ran 3 separate times for 90 seconds each at a 60 Hz tick rate, placing the MRZR vehicle's starting location at a different point on the terrain for each run. The coordinate at the vehicle's center-of-gravity (COG) was used to determine whether or not the vehicle was allowed to be traversing over the terrain beneath it.

Data outlined in Table 3 lists the resulting positive and negative traversals during the simulation. If the NN at the vehicle's COG location at a given simulation step (tick) returned a positive result, it would be counted as a positive traversal, indicating that the vehicle was allowed to be driving over the given coordinate. If it returned negative, it would

Algorithm 1 Waypoint Finder

while Simulation is running **do**
  $r \leftarrow$ Set initial radius of waypoint vector to 1 meter
  $v \leftarrow$ Set initial waypoint vector to -90 degrees
  while Waypoint is not found **do**
    $C \leftarrow$ Get coordinate of world at the end of vector with radius r
    $G \leftarrow$ Get pixel coordinate of gray-scale image wi respect to C
    $A \leftarrow$ Get filtered 3x3 average pixel value at pixel
    $Y \leftarrow$ NeuralNetwork(G) // Put value into NN
    if Y is -1 **then**
      Waypoint is not found
      v = v + 1 // Move 1 degree to the right
      if $v > 90$ degrees **then**
        r = r + 1 meter // After 180 degrees, increase vector radius
      end if
    end if
    if Y equals 1 **then**
      Waypoint is found.
    end if
  end while
  // Waypoint at destination G is valid
  $X \leftarrow$ MRZR location
  while Destination G is not reached **do**
    $\theta \leftarrow$ set steering angle towards G
    Accelerate vehicle $\leftarrow$ PID(current velocity)
    $X \leftarrow$ Update vehicle location
  end while
end while

be counted as a negative traversal, indicating that the vehicle was not allowed to be traveling over the given coordinate. This correlation between positive and negative traversals is also displayed within the bar graph shown in Fig. 11.

TABLE III
SIMULATION RESULTS

Test	Pos. Traversals @ COG (1)	Neg. Traversals @ COG (-1)	Pos. Traversal Rate
1	5362	39	99.28 %
2	5269	132	97.56 %
3	5243	158	97.07 %

VI. CONCLUSIONS & FUTURE WORK

This paper has successfully shown that by training a shallow NN over geo-located aerial terrain image data, an AV can successfully traverse through a virtually simulated off-road environment. The simulation was able to run in real-time within the UE4 game engine utilizing its C++ libraries to integrate the trained NN, which was imported from MATLAB and translated into C++. This streamlined and integrated approach prevented long simulation times as seen in other AV off road simulations.

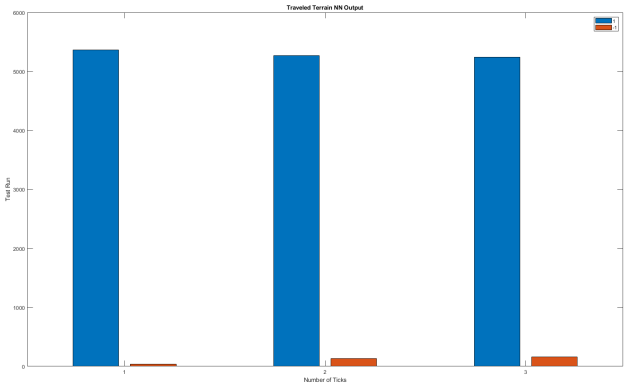

Fig. 11. Simulation Results

The MRZR modeled within UE4 had a high percentage rate during simulation of driving over traversable terrain as seen in Table 3, which stated to be at 97% success rate and above. This leads to a high degree of confidence that the traversed path taken was able to be driven over within the real-world environment the simulation was modeled after at the KRC. A path of the AV's traversal during the simulation could be recorded and translated onto a real-world AV platform driving over the physical terrain for further analysis. This conversion would assume that the simulation is properly initialized by importing up-to-date terrain information to display within the virtual world.

Further improvements to the simulation could be done by also determining if the waypoint being calculated by the simulation returns a positive NN result, while also checking if the waypoint destination's elevation is not too drastically higher or lower than the current AV elevation. This further modification would prevent the simulation from either trying to traverse over a steep cliff/hole, or climb an insurmountable hill. Other edge-cases presented in AV control would have to be considered as well. It is also of note that the vehicle was not placed in any challenging situation during any of the test runs.

This paper therefore has only demonstrated the potential of using NNs for real-time AV control by utilizing human-determined traversable terrain locations seen originally in Fig. 2. Improvements to the NN and its input data could also be done by incorporating data specifically processed as either grass, dirt, mud, water, etc. The NN could then make a better evaluation using a deeper knowledge of the terrain, rather than what was presented as input during the pre-processing of the data shown in this work.

ACKNOWLEDGMENT

I would like the thank all the students and full-time employees at the Keweenaw Research Center for their work in data acquisition regarding the terrain test course point cloud and aerial imagery data used throughout this paper.

Thank you especially to my family and friends, whose continued help and support makes work like this possible.

REFERENCES

[1] G. Alessandretti, A. Broggi and P. Cerri, "Vehicle and guard rail detection using radar and vision data fusion", IEEE Trans. Intell. Transp. Syst., vol. 8, no. 1, pp. 95-105, Mar. 2007.

[2] M. Endsley, "Autonomous Driving Systems: A Preliminary Naturalistic Study of the Tesla Model S.", Journal of Cognitive Engineering and Decision Making, vol. 11, no. 3, pp. 225-238, Feb. 1, 2017,

[3] G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pip, K. Eder, "On Determinism of Game Engines used for Simulation-based Autonomous Vehicle Verification", arxiv, Apr. 7, 2021.

[4] T. Erex, Y. Tassa, E. Todorov, "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE, and PhysX", 2015 IEEE International Conference on Robotics and Automation, May 26-30, 2015.

[5] MATLAB, "Get Started with Deep Learning Toolbox", Mathworks, R2021b, 2021, [Online] Available: https://www.mathworks.com/help/deeplearning/getting-started-with-deep-learning-toolbox.html, [Accessed: Dec. 12, 2021].

[6] T. Pollok, et. al, "UnrealGT: Using Unreal Engine to Generate Ground Truth Datasets", International Symposium on visual Computing, pp. 670-682, Oct. 21, 2019.

[7] D. Talwar, S. Guruswamy, N. Ravipati and M. Eirinaki, "Evaluating Validity of Synthetic Data in Perception Tasks for Autonomous Vehicles," 2020 IEEE International Conference On Artificial Intelligence Testing (AITest), 2020, pp. 73-80.

[8] "Unreal Engine", Epic Games, 2021, [Online] Available: https://www.unrealengine.com/en-US/, [Accessed: Nov. 04, 2021].

[9] Z. Jeffries, C. Majhor, J. Carter, S. Kysar, J. P. Bos, "MUONS path planning performance for a vehicle with complex suspension in Unreal", in Proc. SPIE 11748, Autonomous systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2021. Apr. 12, 2021.

[10] P. Young, S. Kysar, J.P. Bos, "Unreal as a simulation environment for off-road autonomy", in Proc. SPIE 11415, Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2020, May 19, 2020.

[11] "Blender", The Blender Foundation, 2021, [Online] https://www.blender.org/, [Accessed: Nov. 04, 2021]