Open Driving Vision

Ben Gunnels, Timothy Schneider

March 2025

1 Project Description

The goal of this project was to build an application that creates a visual simulation of a car driving down the road and encountering road signs and objects along the shoulder. Additionally, we applied an image segmentation model to the images produced by the simulation for training and testing. A machine learning engineer could use this application to prototype and test an image segmentation model on our driving simulator to detect objects and avoid collisions.

We kept the scope of this project limited to a car driving on a single-lane road. Objects in the car's view included a variety of blank roadway signs, traffic cones, and mileage markers.

2 Simulator Methodology

The front dashed median line measure was 10% of screen height. The back dashed median line was 2% of the screen height. The difference between the starting point of the first median line and the second median line was 27% of screen height. The start of the first median line was 18% of the screen height and the start of the second median line was 36% of the screen height. Using all of the above information, we can calculate the size of a median line given its distance from the origin (where the line emerges from the bottom of the screen) by deriving the linear function.

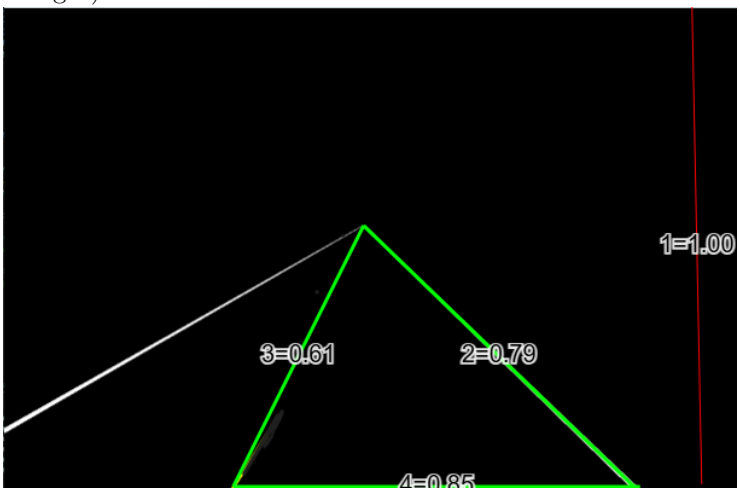
$$y = -0.32x + 0.18 \tag{1}$$

Here, y describes the relative size of the median line and x describes its Euclidean distance from the origin. Similarly, to calculate the length of the gap between dashed lines, the equation becomes:

$$y = -0.32x + 0.2232 \tag{2}$$

Figure 1: Real-life inspiration for the application and measurement of the road median line relative to the photo height.

Figure 3: Measurement of the road median angle to the horizon (size relative to photo height).

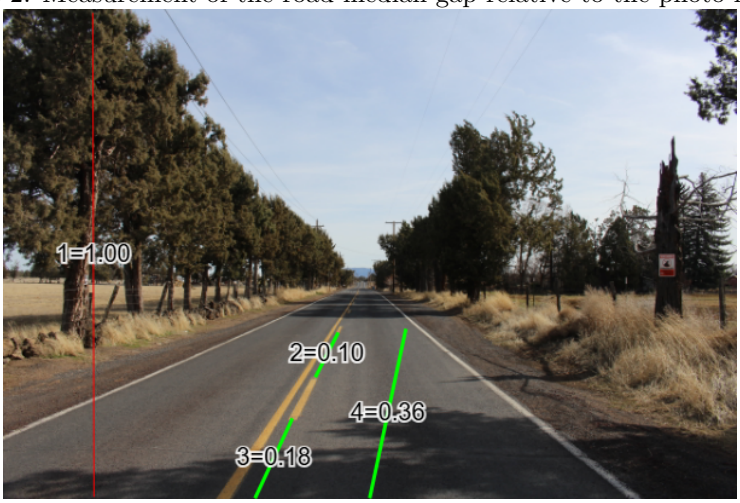


Using Figure 3, the angle of the median line relative to the bottom of the screen was calculated to be 63.018°. This angle, along with the distance from the start of a dashed median line to the origin, was used to calculate the x and y positions that defined each median line.

The x position was calculated as:

$$x = h \cdot \cos(63.018°) \tag{3}$$

Figure 2: Measurement of the road median gap relative to the photo height.



The y position was calculated as:

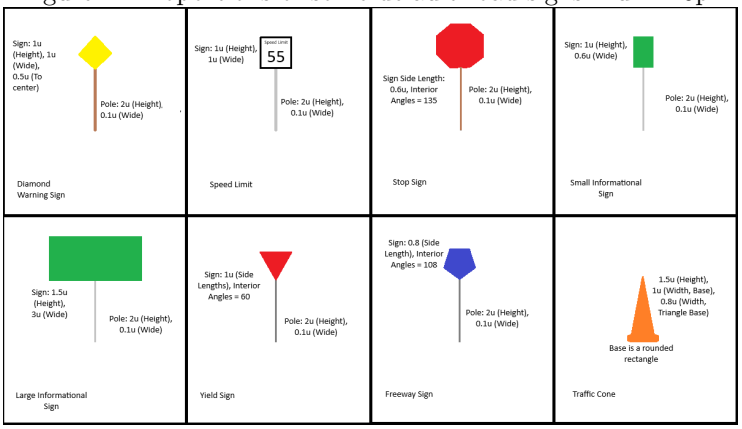
$$y = h \cdot \sin(63.018^\circ) \tag{4}$$

Where h is the Euclidean distance from the origin to the start of the dashed median line. Using Equations (1) and (2) in concert with (3) and (4) allowed us to generate the two points (bottom and top) needed to define a dashed median line along the median path. Equations (1) and (2) generate the size of the dashed median line and gap, respectively, while Equations (3) and (4) provide the correct position in the x-y plane.

To achieve the effect of animation, the median lines are moved as a percentage of the gap that precedes them. The necessary x and y adjustments are then calculated using Equations (3) and (4).

To produce road signs and other roadside objects, each object was sketched with some initial proportions. For most of the objects, a default pole size is placed below the sign at its center. Some objects contain double poles, with one near each edge of the sign. The signs "spawn" in the simulator with a minimum size, usually only a few pixels in area. This gives the simulation the effect of objects organically coming into view from the horizon. The objects are randomly skewed along the horizontal axis relative to the horizon. This allows their trajectories to vary as they move closer to the viewer.

Figure 4: Proportions of some default road signs. 1u = 10px.



The objects we created are a collection of points for both the sign and the pole(s). After each frame, these points are moved proportionally to their distance from the center of the frame's perspective. Or more accurately, the point where the left and right road lines meet in the distance beyond the horizon. This distance value is increased by some user defined rate r, and, given the point's angle relative to the center, the new x and y positions are calculated.

$$d_{\text{new}} = r \times d_{\text{old}} \tag{5}$$

$$x = x_{\text{center}} + d \cdot \cos(\theta) \tag{6}$$

$$y = y_{\text{center}} + d \cdot \sin(\theta) \tag{7}$$

The value of **theta** is computed while spawning the object and is held in radians. This ensures that the proper sign is computed as the object is moved throughout the frames.

Consequently, as the simulation unfolds, the objects appear to grow in size as if the car is moving toward them. The points move away from the horizon but maintain their same angle causing the object to appear to grow as the points drift apart.

The simulator has two main modes: **stream simulator**, and **random simulator**. The stream simulator creates a continuous flow of images giving the perception of a car moving steadily down the road. The randomized simulator is optimized to provide samples of training data for an image model. In the randomized simulator, each frame has objects that are randomly spawned and placed in the image.

3 Image Segmentation Methodology

The thrust of this project was to explore how a machine learning model could be trained on these images as training data. This project allows engineers to create thousands of training samples in a matter of seconds. If a model can be accurately designed and applied to these samples, it would generate a large amount of free, pre-labeled data, providing valuable support for an autonomous vision system. The open nature of this project encourages engineers to be part of the simulation process, creating custom objects and adding visual filters for their own purposes.

The model applied to the simulation was a U-Net, which consisted of 5 down-sample layers and 4 up-sample layers (see Figure 5). The final layer of the model was a 2D convolutional layer that output the predictions to the correct class labels. Conceptually, the model consists of a contracting set of layers and an expanding set of layers. The contracting path serves as feature extraction, wherein it distills the important information in the image related to the labels. The expanding path regenerates the masked features in their original space so that the mask can be mapped onto the original image. For a more thorough overview of the U-Net model, see Aramendia [1].

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
functional (Functional)	[(None, 64, 64, 96), (None, 32, 32, 144), (None, 16, 16, 192), (None, 8, 8, 576), (None, 4, 4, 320)]	1,841,984	input_layer_1[0][0]
sequential (Sequential)	(None, 8, 8, 512)	1,476,608	functional[0][4]
concatenate (Concatenate)	(None, 8, 8, 1088)	0	sequential[0][0], functional[0][3]
sequential_1 (Sequential)	(None, 16, 16, 256)	2,507,776	concatenate[0][0]
concatenate_1 (Concatenate)	(None, 16, 16, 448)	0	sequential_1[0][0], functional[0][2]
sequential_2 (Sequential)	(None, 32, 32, 128)	516,608	concatenate_1[0][0]
concatenate_2 (Concatenate)	(None, 32, 32, 272)	0	sequential_2[0][0], functional[0][1]
sequential_3 (Sequential)	(None, 64, 64, 64)	156,928	concatenate_2[0][0]
concatenate_3 (Concatenate)	(None, 64, 64, 160)	0	sequential_3[0][0], functional[0][0]
conv2d_transpose_4 (Conv2DTranspose)	(None, 128, 128, 24)	34,584	concatenate_3[0][0]

Total params: 6,534,488 (24.93 MB)
Trainable params: 4,690,584 (17.89 MB)
Non-trainable params: 1,843,904 (7.03 MB)

Figure 5: U-Net Architecture.

Since the model is in its infancy, it exhibits high variance or bias, as seen in Figures 6 and 7. This highlights the need for further refinement through data preprocessing, feature engineering, and hyperparameter optimization to improve it's performance. Then next steps involve iterating on the model– generating new training images fine-tuning confidence thresholds, and adjusting the minimum pixel read for a class. These refinements, along with experimenting with deep learning methods, help determine whether the model holds promise.

Figure 6: Early-stage model prediction mask compared to the ground truth mask.

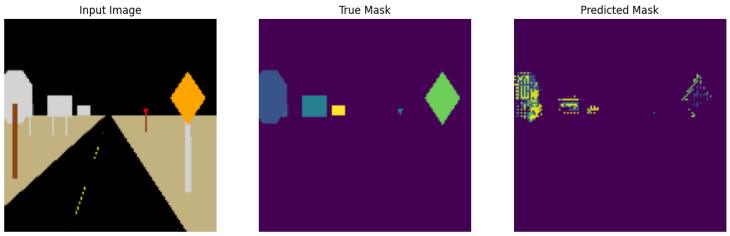
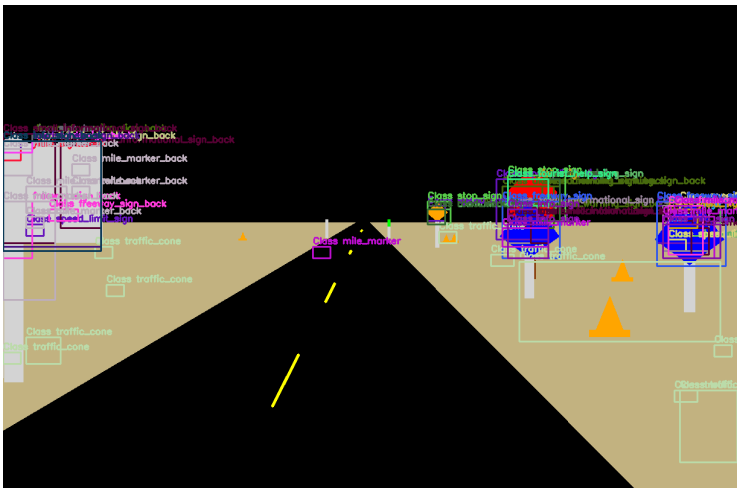


Figure 7: Visualization of the early-stage model with live masks overlaid on images.



The model was trained for over 200 epochs with 5 validation sub-splits, 2 validations steps, and 7 steps per epoch. Batches of training images were randomized every 40 epochs which allowed for diverse sampling. The terrains of the image were randomly assigned by the simulator to be one of grass, clay, sand, or rock.

The final model, as shown in Figures 8 and 9, has undergone significant refinement and demonstrates improved performance with reduced variance and bias. Data pre-processing, feature engineering, and hyperparameter optimization have been successfully applied to enhance its accuracy and reliability. Iterative improvements, continuous training, fine-tuning of confidence thresholds, and adjusting pixel requirements for each class have contributed to more robust predictions, showcasing its potential to meet or exceed expectations in real-world applications.

The model was tested on a road image stream simulation. The stream simulation consisted of 500 frames with a grass terrain. In the stream simulation,

road objects spawned more irregularly than they did in the training images. This provided for a more realistic expectation of a driving experience. The model showed tremendous promise, as expected, in correctly classifying the roadsigns. With more training and model tuning, the performance would be commercial quality.

Figure 8: Final trained model predictions with corresponding masks.

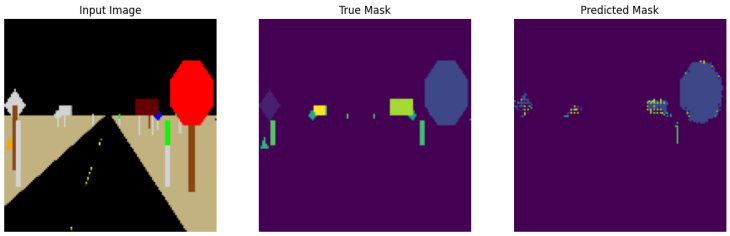
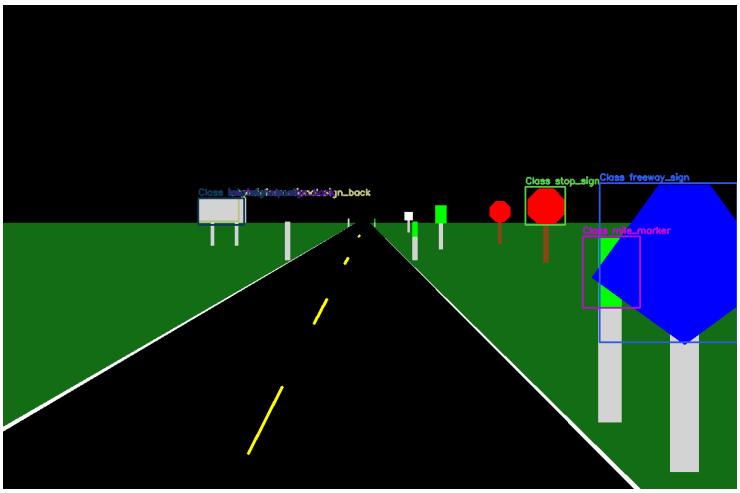


Figure 9: Visualization of the final trained model with live masks overlaid on images.



4 Final Thoughts

The Open Driving Vision simulator shows promise in its goal of creating random driving image data for use in autonomous driving machine learning models. We were able to apply a basic U-Net model to demonstrate the image segmentation task as an example use case. We recommend that, in the future, more objects be added to expand the number of classes that the model must detect. For example, cars moving in the oncoming lane, or cars in the driver's lane. Additionally, providing objects with dynamic shading, multiple colors,

and variable lighting conditions would be worthwhile additions to help the data become more applicable to real-world conditions.

Source code can be found at [3], and [4].

5 References

[**1**] Ito Aramendia, A. (2024, January 31). Decoding the U-Net: A complete guide. Medium. Retrieved March 24, 2025 from https://medium.com/@alejandro.itoaramendia/decoding-the-u-net-a-complete-guide-810b1c6d56d8

[**2**] Image segmentation: Tensorflow Core. TensorFlow. (2024, August 16). https://www.tensorflow.org/tutorials/images/segmentation

[**3**] https://github.com/ben-gunnels/Open-Driving-Vision

[**4**] https://github.com/T1mSchneider/Open-Driving-Vision