Synthetic Data for Vision-Based Lane and Stripe Segmentation

Lukas Rennhofer*, **Matteo Prader***
Dominik Pascher, Máté Salánki, Maximilian Panzenböck, Moritz Rottensteiner
Höhere Technische Bundeslehr- und Versuchsanstalt Wiener Neustadt
Department of Computer Science
2700 Wiener Neustadt, Austria
*Corresponding author's email: rennhofer@tracer.team
prader@tracer.team

***Abstract*—Training robust vision systems for autonomous navigation requires large annotated datasets, but collecting and labeling real-world data is costly. This paper presents a scalable pipeline for generating synthetic datasets with stripe segmentation for robotic navigation. A procedural renderer produces diverse road scenes with automatically generated pixel-accurate segmentation masks, enabling large-scale supervised training without manual annotation. Several U-Net–based segmentation models are trained and evaluated to analyze performance–complexity trade-offs. Experiments show effective simulation-to-real transfer, with the best model achieving an intersection over union (IoU) of 0.89 on synthetic data and 0.73 on real images from a real-world competition environment. Ablation studies indicate that modeling motion blur improves transfer performance, while runtime tests demonstrate that lightweight models enable near real-time inference on embedded robotic platforms.**

***Index Terms*—Autonomous navigation, synthetic data, robotics, simulation**

I. Introduction

This paper presents an approach for automated stripe detection based on a procedurally generated synthetic dataset. A dedicated rendering pipeline is used to generate randomized scenes containing striped planar surfaces under varying lighting conditions and camera configurations, each paired with a pixel-accurate ground-truth segmentation mask (see Figure 1).

Using this synthetic data, a deep learning model is trained to perform stripe segmentation in high-resolution images. The proposed approach is motivated by robotic navigation tasks such as lane identification, as encountered in autonomous driving systems. The paper further investigates the extent to which models trained exclusively on rendered data transfer to real-world imagery.

II. Synthetic Dataset Generation

A. Renderer Architecture

To create large datasets for road markings and lane detection, a synthetic rendering system was developed with a focus on speed and flexibility. For each rendered frame, the system also generates a precise ground-truth mask, eliminating manual pixel-level labeling. The renderer is based on the OpenGL Mesa driver with GPU support [1]–[3]. A custom seed-management framework ensures strict reproducibility: each generated scene can be reconstructed exactly from its logged seed. The pipeline supports multiple output resolutions and multi-threaded rendering, enabling efficient large-scale dataset generation while preserving deterministic behavior for debugging and controlled experiments.

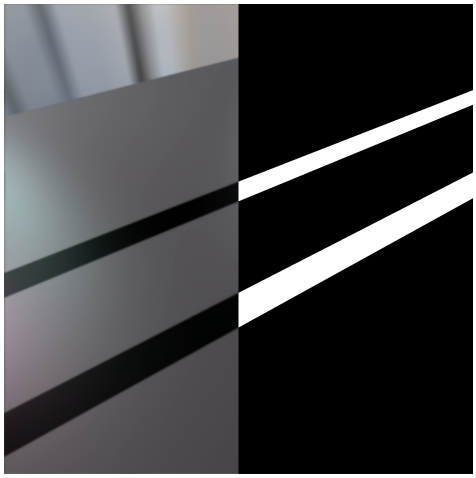

Fig. 1: Rendered road scene (left) and the corresponding pixel-level ground-truth mask (right).

B. Procedural Scene Generation

Instead of modeling every road and lane marking manually, scenes are generated procedurally. Road textures and stripe patterns are synthesized directly in shader programs, which allows fine control over geometry and appearance. Key parameters such as stripe width, spacing, orientation, and surface texture are randomized within defined bounds. This creates diverse scenes while maintaining realistic structure. A central advantage of this pipeline is automatic pixel-level annotation: every rendered image is paired with an exact ground-truth mask, which enables direct use in supervised training and quantitative evaluation. Figure 1 shows this one-to-one correspondence.

C. Scene Randomization and Variability

To ensure both diversity and reproducibility, scene generation uses controlled stochastic variation over a fixed set of renderer parameters. These parameters affect lane geometry, stripe appearance, camera pose, ground-plane dimensions, and illumination. Each scene is determined by a deterministic global seed; however, reproducibility across systems also requires explicit specification of all parameter domains. The scene configuration is formalized as a random parameter vector as shown in Eq. 1.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_{\text{lane}} & \theta_{\text{stripe}} & \theta_{\text{camera}} & \theta_{\text{plane}} & \theta_{\text{lighting}} \end{bmatrix}, \tag{1}$$

where θ_{lane} groups parameters controlling lane geometry, θ_{stripe} defines the appearance of lane markings, θ_{camera} encodes camera position, orientation, and intrinsics, θ_{plane} specifies the dimensions of the ground plane, and θ_{lighting} captures illumination-related properties. Each group $\theta_{\cdot}$ consists of one or more scalar renderer parameters and is sampled independently from bounded distributions,

$$\theta_i \sim \mathcal{D}_i(\theta_i^{\min}, \theta_i^{\max}),$$

where $\mathcal{D}_i$ denotes a parameter-specific distribution (e.g., continuous uniform, discrete uniform, or Bernoulli), depending on the parameter type. Under the assumption of independence between parameter groups, the joint distribution of scene configurations factorizes as

$$p(\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\theta_i). \tag{2}$$

where N denotes the total number of parameter groups, i.e., the cardinality of the set of parameter types

$$\mathcal{N} = \{\text{lane}, \text{stripe}, \text{camera}, \text{plane}, \text{lighting}\}.$$

For each rendered frame, the renderer logs the global random seed, which deterministically defines the complete parameter vector $\boldsymbol{\theta}$. Given the same seed and identical parameter bounds, the scene is reproduced exactly.

Table I summarizes all randomized renderer parameters used in this work, together with their sampling distributions and numerical ranges.

In addition to nominal operating conditions, the renderer can generate *extreme scenes*. This mode is controlled by the boolean parameter `isExtreme` and a user-defined ratio of extreme samples in the dataset. When enabled, sampling is biased toward parameter-range boundaries, increasing the frequency of challenging conditions such as strong camera tilt, uneven illumination, high roughness, strong blur, and atypical stripe geometry.

D. Domain Gap and Limitations

Despite the benefits of synthetic data generation, a domain gap between rendered scenes and real competition environments remains unavoidable. The current renderer cannot fully reproduce all physical and visual effects observed during real runs. In particular, different ground materials can only be approximated and are not simulated in a physically accurate way. Furthermore, transparent and glass-like objects are not considered, as their simulation would be computationally too expensive.

TABLE I: Scene randomization parameters and sampling ranges

Parameter	Distribution	Range / Values
laneWidth	Uniform	$[0.08, 0.20]$ m
stripeThickness	Uniform	$[0.005, 0.03]$ m
dashLength	Uniform	$[0.2, 1.2]$ m
dashSpacing	Uniform	$[0.1, 0.8]$ m
gaussianBlur	Uniform	$[0.0, 1.0]$
motionBlur	Uniform	$[0.0, 1.0]$
motionBlurAngle	Uniform	$[0°, 180°]$
lensDistortion	Uniform	$[-0.08, 0.08]$
shadowStrength	Uniform	$[0.0, 0.7]$
shadowScale	Uniform	$[0.02, 0.15]$
shadowShapeCount	Discrete uniform	$\{2, \ldots, 10\}$
shadowShapeRadius	Uniform	$[0.5, 2.5]$ m
shadowShapeSoftness	Uniform	$[0.2, 0.9]$
roughness	Uniform	$[0.0, 0.6]$
ambientIntensity	Uniform	$[0.3, 1.2]$
camPosX	Uniform	$[-1.5, 1.5]$ m
camPosY	Uniform	$[-1.0, 1.0]$ m
camPosZ	Uniform	$[0.8, 1.6]$ m
camYaw	Uniform	$[-20°, 20°]$
camPitch	Uniform	$[-15°, 15°]$
camRoll	Uniform	$[-5°, 5°]$
camFov	Uniform	$[50°, 90°]$
planeWidth	Uniform	$[4, 10]$ m
planeLength	Uniform	$[6, 20]$ m
numLights	Discrete uniform	$\{1, 2, 3\}$

Potential mitigation strategies: A practical mitigation strategy is to extend domain randomization with additional ground textures and roughness profiles to better approximate real surfaces. Another option is a physically based rendering (PBR) pipeline, which could model realistic materials, including transparent objects; however, this would substantially increase rendering cost and dataset generation time, and is therefore not adopted here in order to maintain a computationally efficient and scalable data generation pipeline.

III. Stripe Detection Model

This section describes the learning-based stripe detection model trained on the synthetic dataset introduced in Section II. The following subsections outline the network architecture and prediction objective, describe the training procedure, and analyze multiple scaled model variants to assess performance–complexity trade-offs. The model architecture is grounded in principles of semantic segmentation [4].

A. Model Overview

The stripe detection model is based on a U-Net convolutional neural network, which is well suited for pixel-wise image segmentation tasks due to its encoder–decoder structure [5]. The encoder progressively captures high-level contextual information through convolutional layers with batch normalization [6], while the decoder restores spatial resolution to enable precise

localization of stripe regions. The network is implemented using the TensorFlow framework [7] with the Keras high-level API [8].

The model takes a 256×512 RGB image as input and produces a dense probability map indicating the likelihood of stripe presence at each pixel location. Figure 2 illustrates a representative output, where brighter regions correspond to higher predicted stripe probabilities.

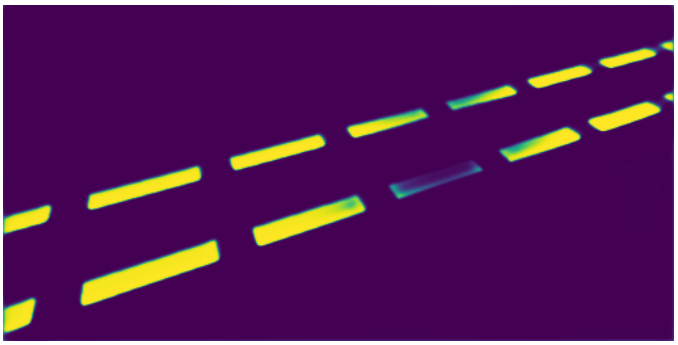

Fig. 2: Probability heatmap output of the stripe detection model, where brighter regions indicate a higher likelihood of stripe presence.

Segmentation performance is evaluated using the Intersection over Union (IoU) metric [9], which measures the spatial overlap between the predicted segmentation mask $\hat{M}$ and the ground-truth mask M:

$$\mathrm{IoU}(\hat{M}, M) = \frac{|\hat{M} \cap M|}{|\hat{M} \cup M|}. \tag{3}$$

Additionally, Precision, Recall, and F1-Score are reported to provide a more detailed assessment:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad \mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4}$$

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}, \tag{5}$$

where TP denotes true positives (correctly predicted stripe pixels), FP denotes false positives (background pixels incorrectly classified as stripes), and FN denotes false negatives (stripe pixels incorrectly classified as background). These metrics are bounded between 0 and 1, with higher values indicating better performance.

B. Training Strategy

All models are trained using a common optimization setup to ensure comparability across architectural variants. The synthetic dataset is split into disjoint training, validation, and test subsets, with approximately 70–80% of the samples used for training and the remainder reserved for validation and evaluation. Model optimization is performed using mini-batch stochastic gradient descent with the Adam optimizer [10]. Training minimizes the binary cross-entropy (BCE) loss between the predicted probability map $\hat{Y}$ and the ground-truth mask Y:

$$\mathcal{L}_{\mathrm{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} \left[Y_i \log \hat{Y}_i + (1 - Y_i) \log(1 - \hat{Y}_i) \right], \tag{6}$$

where N denotes the number of pixels. Optimization is performed using this loss, while segmentation quality is evaluated using the IoU metric defined in Eq. 3. Since stripe pixels are sparse compared to background pixels, the dataset exhibits class imbalance. Binary cross-entropy may bias predictions toward the background class; alternatives such as weighted BCE or Dice-based losses could mitigate this effect. In the conducted experiments, standard BCE was sufficient and did not significantly affect convergence or final IoU. Training progress is monitored using training and validation loss curves, where a consistent decrease indicates convergence and divergence suggests overfitting.

TABLE II: Comparison of trained model variants

Model	Batch	Epochs	Dataset	LR	IoU	F1	Params
alpha	16	20	12k	1×10^{-3}	0.71	0.79	0.9M
beta	16	30	12k	1×10^{-3}	0.75	0.82	1.1M
gamma	16	40	16k	5×10^{-4}	0.80	0.86	1.8M
delta	8	50	20k	5×10^{-4}	0.83	0.88	3.2M
atlas	8	60	32k	3×10^{-4}	0.85	0.90	4.6M
hermes	32	25	12k	1×10^{-3}	0.78	0.84	0.6M
apollo	8	70	32k	3×10^{-4}	0.88	0.92	6.1M
athena	16	50	24k	5×10^{-4}	0.86	0.91	2.9M
zeus	8	80	32k	3×10^{-4}	0.89	0.93	7.4M
omega	16	60	32k	3×10^{-4}	0.88	0.92	5.2M

C. Model Variants and Comparative Analysis

To analyze the trade-off between segmentation accuracy and model complexity, multiple scaled variants of the stripe detection network were trained. The naming scheme is inspired by symbolic model naming conventions commonly used in the literature, including those employed in Google DeepMind's Alpha-series models [11]. All variants were trained on the same synthetic dataset using identical data splits and optimization settings to ensure fair comparison. Training was performed on a desktop GPU using the Adam optimizer, with learning rates (LR) selected based on model scale and training stability and either kept constant or gradually reduced during training. During inference, predicted probability maps were converted into binary segmentation masks using a fixed threshold of $\tau = 0.5$. An overview of the training configurations and achieved performance for all model variants is provided in Table II.

Model capacity correlates with segmentation quality: smaller models such as Hermes achieve moderate IoU with fewer parameters, while larger variants like Apollo and Zeus achieve higher IoU at increased complexity, at the cost of higher overfitting risk. To investigate the domain gap, models trained on synthetic data were fine-tuned on 50 real images, reducing IoU from 0.73 to 0.71, likely due to overfitting. This indicates that small-scale fine-tuning is insufficient, while larger datasets or improved adaptation strategies may still be beneficial. Figure 3 shows the trade-off between IoU and model complexity,

where Pareto-optimal models provide the best balance between performance and parameter count.

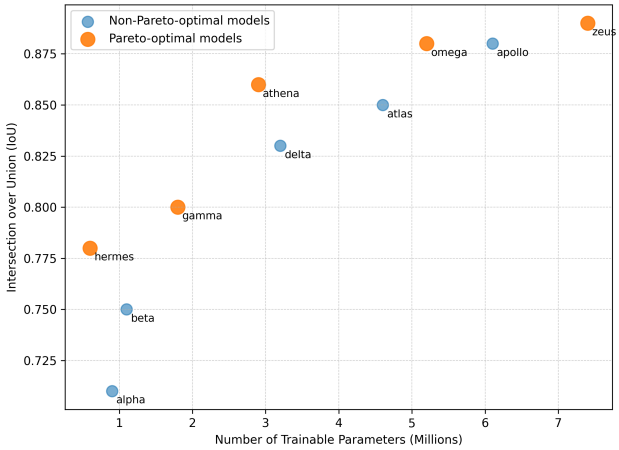

Fig. 3: Trade-off between model complexity and IoU across evaluated variants.

IV. Experimental Results and Analysis

This section evaluates stripe segmentation models trained on synthetic data, examining performance on both rendered and real-world inputs as well as embedded robotic hardware using consistent preprocessing and inference settings.

A. Experimental Setup

The experimental evaluation investigates whether models trained exclusively on synthetic data generalize to real competition environments under practical computational constraints. Stripe detection is formulated as binary semantic segmentation, where the model predicts a stripe mask from an RGB image. All images are resized to 512×256 pixels, and the same preprocessing pipeline (resizing and normalization) is applied in all experiments. Evaluation is performed on two data sources:

- A synthetic test set generated using unseen random seeds.
- A real-world dataset of 500 images captured from an official competition game table. Binary masks were annotated in *labelme* [12] using a semi-automatic *SmartEdit* style plugin. Annotation took about 4 hours in total. Minor boundary inaccuracies remain, with an estimated deviation of 1–2 pixels, and manual correction was required for roughly 10% of the images.

B. Qualitative Evaluation

Qualitative evaluation analyzes segmentation behavior under domain shift and identifies typical failure modes. Errors mainly arise from limitations in physically accurate material simulation, scene objects, and visual effects that cannot be fully represented in the synthetic domain, despite modeling non-uniform illumination, motion blur, and sensor noise. For an ablation-style evaluation, the Zeus model was retrained with modified simulation settings to assess individual rendering effects. Training with all effects enabled yields the best real-world performance (IoU 0.73). Removing motion blur reduces IoU to 0.69 (∼6% decrease), while disabling shadows and lens distortion results in IoUs of 0.71 and 0.72, respectively, indicating that motion blur contributes most to sim-to-real transfer. Figure 4 shows representative predictions of the **Zeus** model and demonstrates that stripe structures learned from synthetic data transfer to real scenes despite increased visual complexity.

Quantitatively, Zeus achieves an IoU of 0.89 ± 0.03 on synthetic data and 0.73 ± 0.07 on real-world images; Hermes reaches 0.78 ± 0.05 and 0.59 ± 0.10 under the same conditions. The higher variance on real-world images reflects domain shift effects.

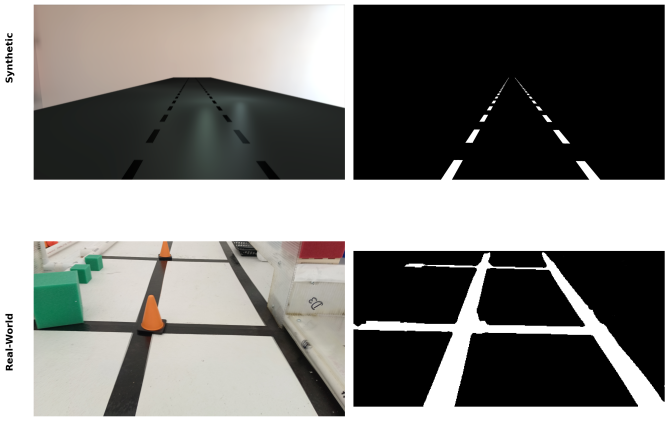

Fig. 4: Predicted segmentation masks for a synthetic test image (top) and a real-world competition image (bottom) using **Zeus**.

Figure 5 shows the IoU distribution across both models and domains. Each box displays median, interquartile range, and min/max whiskers. Zeus consistently outperforms Hermes, with higher variance on real data.

Fig. 5: IoU distribution for Zeus and Hermes models on synthetic and real data.

C. Typical Failure Cases

Figure 6 shows representative failure cases of the Zeus model on real-world images. The overlay highlights high-confidence predictions (yellow), corresponding to probabilities above a fixed threshold, while lower-confidence regions are not visualized. Both examples contain a stripe surrounded by a transparent, reflective surface. The model produces unstable predictions because such material properties are not represented in the synthetic dataset. Notably, predictions are concentrated along stripe edges, which likely exhibit stronger contrast and are therefore learned more reliably than uniform interior regions. Changing the camera pose (right) does not resolve the issue, indicating a systematic domain gap caused by simplified material simulation, as discussed in Section II-D.

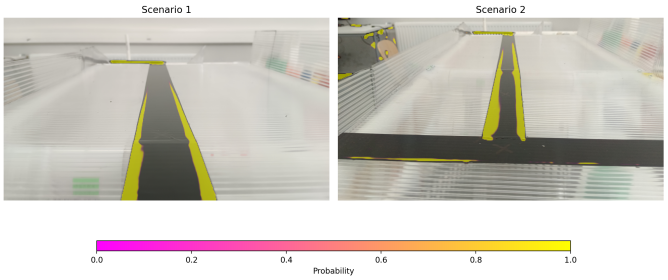

Fig. 6: Failure cases of the Zeus model. Yellow overlay indicates high-confidence stripe predictions (above threshold). Left: reflective ground causes fragmented edge-focused detections. Right: zoomed-out view of the same environment.

D. Runtime Performance

To evaluate deployment feasibility on robotic hardware, inference runtime is measured on both a desktop system and an embedded platform. As a reference, inference is performed on a desktop GPU. On this system, the evaluated models achieve an average inference time of approximately 4–6 ms per frame, corresponding to frame rates well above real-time requirements.

For embedded evaluation, selected model variants are deployed on a Raspberry Pi 3B [13]. All measurements include image preprocessing and model execution. On this platform, lightweight model variants achieve inference times of approximately 180–250 ms per frame, corresponding to frame rates of 4–5 fps. Larger models exhibit inference times exceeding 400 ms per frame and are therefore less suitable for small-scale, time-critical robotic control.

V. Conclusion

This work presents a scalable pipeline for training stripe segmentation models with procedurally generated synthetic data. Controlled scene randomization and U-Net-based segmentation enable fully annotated training data without manual labeling. The best-performing model reached an IoU of 0.89 ± 0.03 on synthetic data and 0.73 ± 0.07 on real images, indicating meaningful sim-to-real transfer. The experiments also show a clear trade-off between model capacity and inference cost, with lightweight variants better suited for embedded deployment. Ablation results indicate that motion blur modeling has a strong impact on transfer quality. Remaining failures are mainly linked to material properties not represented in simulation. The rendering system could adopt physically based rendering to better simulate complex materials, particularly transparent and reflective surfaces that currently limit performance. Additionally, learned domain randomization strategies could adapt parameter distributions based on validation performance to reduce the domain gap more effectively. The procedural generation approach is applicable to other navigation tasks, enabling end-to-end training of complete perception systems in simulation.

Acknowledgment

The authors would like to thank Dr. Michael Stifter for his helpful support and feedback during the development of this project. The authors also thank the members of robo4you for their technical input and support during testing and evaluation.

Code Availability

The source code for the renderer and the training pipeline is publicly available at:

https://git.robo4you.at/TRACER/SDVS

References

[1] Khronos Group, “Opengl – the industry standard for high performance graphics,” https://www.khronos.org/opengl/, 2024, accessed: 2026-01-20.
[2] Mesa Project, “Mesa 3d graphics library,” https://www.mesa3d.org/, 2024, accessed: 2026-01-20.
[3] Khronos Group, “Opengl shading language (glsl) specification,” https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language, 2024, accessed: 2026-01-20.
[4] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3431–3440, 2015, accessed: 2026-01-20.
[5] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” https://arxiv.org/abs/1505.04597, 2015, arXiv preprint.
[6] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” International Conference on Machine Learning (ICML), 2015, accessed: 2026-01-20.
[7] Google, “Tensorflow: An end-to-end open source machine learning platform,” https://www.tensorflow.org/, 2024, accessed: 2026-01-20.
[8] Keras Team, “Keras: Deep learning for humans,” https://keras.io/, 2024, accessed: 2026-01-20.
[9] PyTorch Documentation, “Intersection over union (iou) for image segmentation,” https://docs.pytorch.org/ignite/generated/ignite.metrics.IoU.html, 2024, accessed: 2026-01-20.
[10] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” International Conference on Learning Representations (ICLR), 2015, accessed: 2026-01-20.
[11] Google DeepMind, “Alphafold: Structure prediction,” https://deepmind.google/science/alphafold/, 2024, accessed: 2026-02-17.
[12] Kentaro Wada, “Labelme: Image polygonal annotation with python,” https://github.com/wkentaro/labelme, 2024, accessed: 2026-02-21.
[13] Raspberry Pi Foundation, “Raspberry pi 3b specifications,” https://www.raspberrypi.com/products/raspberry-pi-3-model-b/, 2024, accessed: 2026-01-20.