RBE549: Project 3 - Einstein Vision

Shreyas Devdatta Khobragade
MS in Robotics Engineering
Worcester Polytechnic Institute
skhobragade@wpi.edu
Using 6 late days

Neel Girish Bahadarpurkar
MS in Robotics Engineering
Worcester Polytechnic Institute
nbahadarpurkar@wpi.edu
Using 6 late days

I. Introduction

This project focuses on creating advanced visualizations inspired by Tesla's latest dashboard systems, with the goal of improving human-robot interaction (HRI) in self-driving cars. Visualization systems help to create trust between humans and autonomous machines by providing intuitive insights into how the robot perceives and responds to its surroundings. These visualizations leverage sensory data to assist humans comprehend the robot's decision-making process, making interactions more efficient and transparent. The project entails developing a rendered video demonstrating key capabilities such as lane detection, vehicle classification, pedestrian identification, traffic signals, and road signs, with more advanced functionalities added in stages.

This project is significant because it has the potential to address important HRI concerns for self-driving cars. Self-driving cars utilize complex algorithms to travel safely, but their decisions can be unclear to consumers. By providing real-time information in an intelligible and visually appealing way, the visualization system bridges the gap between technological complexity and human comprehension. This not only helps with debugging during development, but it also builds trust in autonomous systems, opening the path for more widespread use of self-driving technologies.

II. Phase 1 : Phase 1: Basic Features

A. Lane Detection

In our technique, we used a Mask R-CNN [9] lane recognition system to generate precise segmentation maps and identify lane sections from road data. The discovered lane data, including spatial coordinates, was then formatted in JSON, allowing for easy connection with Blender for sophisticated visualization. Custom Python utilities were developed in Blender to convert these world coordinates into Bezier curves; these curves serve as the foundation for generating lane geometries, with array and curve modifiers used to meticulously render lane markings with properties tailored to distinguish between solid and dashed lanes. This technology not only improves the visual realism of lane representations in simulation settings, but it also supports the rigorous geometry analysis required for self-driving cars.

Fig. 1. Vehicle Detection using YOLO v11

B. Vehicle Detection

We used YOLO v11 [1], a cutting-edge object identification framework noted for its excellent accuracy and real-time performance, to accurately identify and localize vehicles inside each frame. The detection algorithm produced detailed bounding box information and vehicle classifications, which were then systematically saved in a JSON file to facilitate subsequent rendering. Using this JSON data, Blender was used to accurately render vehicle models based on the spatial and temporal requirements of each frame, ensuring that the visualizations were high fidelity and closely matched the dynamic traffic scenarios encountered in autonomous driving environments. The vehicle detection output is shown in figure 1.

C. Pedestrian

For pedestrian recognition, we used YOLO v5 [3], which was tuned for the unique job of recognizing and localizing pedestrians in dynamic situations. The model produced detailed results, such as bounding box coordinates and confidence scores for each observed pedestrian. This data was carefully saved in a JSON file for easy interaction with our rendering pipeline. Using the JSON data, Blender was used to precisely display pedestrians in the simulation environment, ensuring precise spatial location. This method improves the

Fig. 2. Pedestrian Detection using YOLO v5

Fig. 3. Traffic Light Detection using Detic

visual accuracy of the simulation, resulting in an accurate representation of pedestrian that are critical for autonomous driving systems. The pedestrian detection output is shown in figure 2.

D. Traffic Light Detection

In our traffic light detection pipeline, we used Detic [2] —a powerful object detection framework—to successfully identify candidate traffic lights in a variety of environmental circumstances. Detic generated initial bounding box coordinates for each detected candidate, which were further adjusted with a unique post-processing technique. This pipeline required converting the selected regions to the HSV color space and using adaptive thresholding to distinguish the lighted patches from the background. Morphological procedures such as erosion and dilation were also used to reduce noise, and pixel intensity histogram analysis allowed for precise classification of traffic light color (red, yellow, or green) despite varied lighting circumstances.

To make interaction with our rendering workflow easier, we aggregated the refined detection outputs—including bounding box coordinates, confidence scores, and calculated color states—into a consistent JSON format. This JSON file acted as an intermediary data format, allowing our bespoke Blender scripts to appropriately simulate traffic lights in the simulation environment. By processing the JSON data, Blender was able to dynamically position and visually portray each traffic signal with accurate spatial alignment and color fidelity, thus boosting the realism and dependability of the autonomous driving display. The output for traffic light detection using Detic is shown in Figure 3.

E. Stop Sign Detection

For stop sign detection again we used YOLO v11 [1], a cutting-edge object detection framework known for its accuracy and real-time performance. The detection findings were routinely saved in a JSON file that comprised spatial data and

Fig. 4. Stop Sign Detection using YOLO v11

confidence scores to ensure consistency and accuracy during later processing. This JSON file was then used in Blender to create stop signs for the simulated area. Our bespoke Blender scripts used the saved data to precisely position and scale the stop signs based on their observed placements in the scene. This method ensured that the generated stop signs were both visually and contextually accurate, resulting in a high-fidelity portrayal of traffic conditions important for autonomous driving systems.The stop sign detection output is shown in figure 4.

F. Depth Map for Accurate Rendering

To accurately position items in the 3D world, we used depth maps generated by the Depth Anything V2 model [5], which enables reliable monocular depth estimation. The bounding box information for each object, obtained from our detection models and saved in JSON files, was utilized to calculate the object's center point within the frame. This center point was then probed against the depth map to determine the

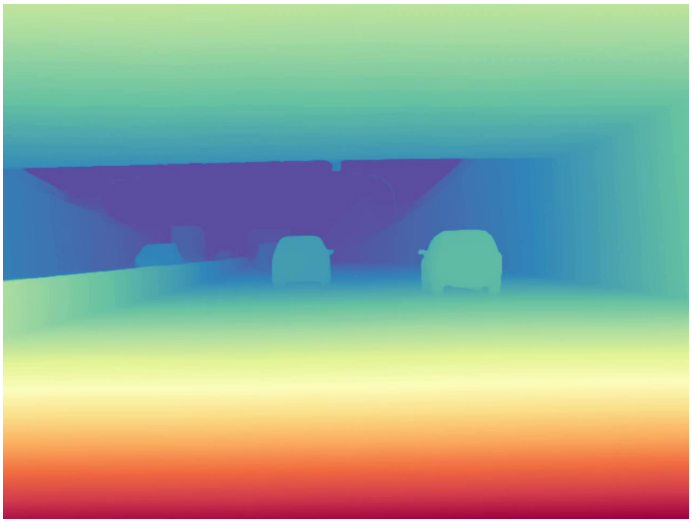

Fig. 5. Depth Map Visualization

specific depth value at that spot. Using this depth information and camera intrinsic matrix values, we calculated the 3D coordinates of each object to ensure proper placement in the simulated environment.

The following formulas were used to translate 2D image coordinates into 3D world coordinates:

$$x_{3d} = \frac{(\text{center}_x - c_x) \cdot \text{scaled\_depth}}{f_x}$$

$$y_{3d} = \frac{(\text{center}_y - c_y) \cdot \text{scaled\_depth}}{f_y}$$

$$z_{3d} = \text{scaled\_depth}$$

Here:

- center_x and center_y are the center coordinates of the bounding box in the image.
- c_x and c_y are the principal point offsets from the camera intrinsic matrix.
- f_x and f_y are the focal lengths in the x and y directions, respectively.
- scaled_depth is the depth value obtained from the depth map at the center of the bounding box.

The depth map visualization is displayed in Figure 5
The rendered outputs are displayed below:

III. Phase 2: Phase 2: Advanced Features

A. Vehicle Pose Detction

To accomplish reliable identification, classification, and orientation estimate for vehicle poses, we used a multi-model pipeline that combined Detic[2], YOLO-World [7], and YOLO-3D []. Detic was chosen as the primary detection model because of its ability to identify cars with high accuracy across a wide range of settings, using its huge vocabulary and excellent localization capabilities. Once vehicles were recognized, their bounding boxes were cut from the original

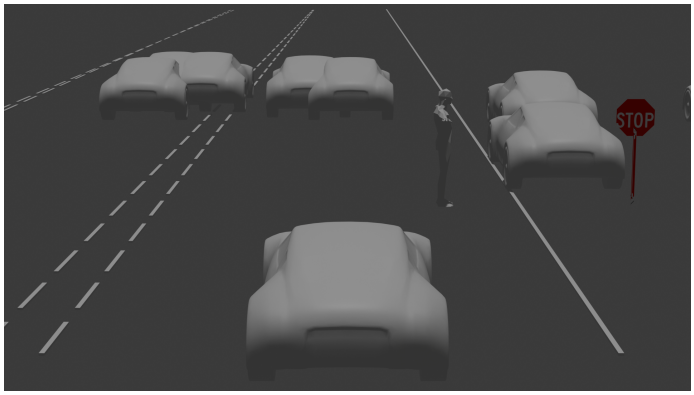

Fig. 6. Output 1

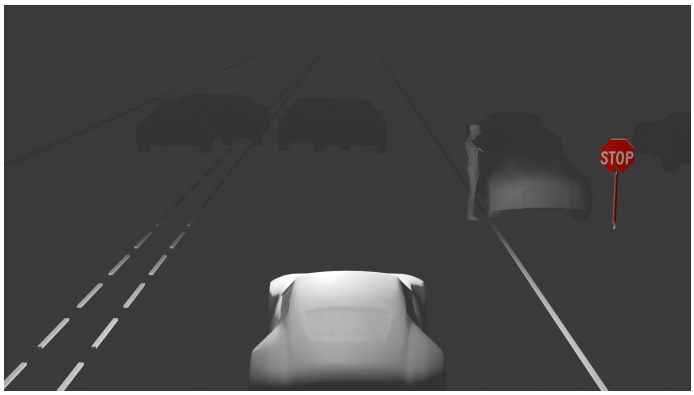

Fig. 7. Output 2

image and sent to YOLO-World, which was programmed with a bespoke vocabulary. This stage allowed for detailed classification of the recognized vehicles into specific groups such as sedan, suv, pickup truck, bicycle, motorcycle and trucks. Following classification, YOLO-3D was used to estimate each vehicle's 3D orientation, ensuring that the spatial alignment and rotational position were correctly captured.

The results of each stage of this pipeline, such as bounding box coordinates, vehicle classifications, and orientation data,

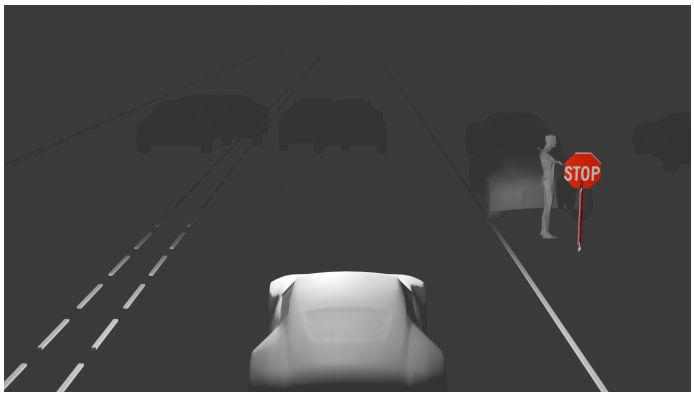

Fig. 8. Output 3

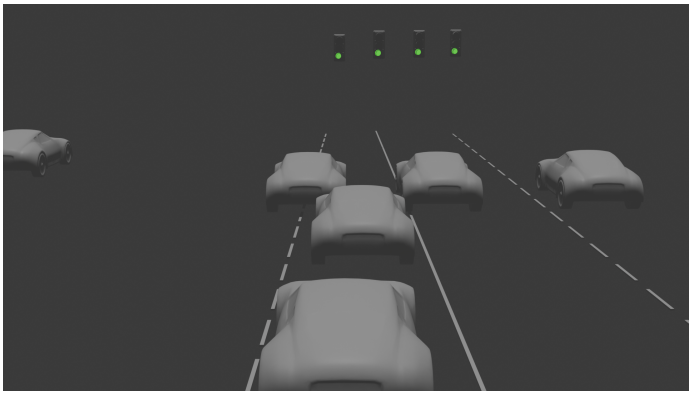

Fig. 9. Output 4

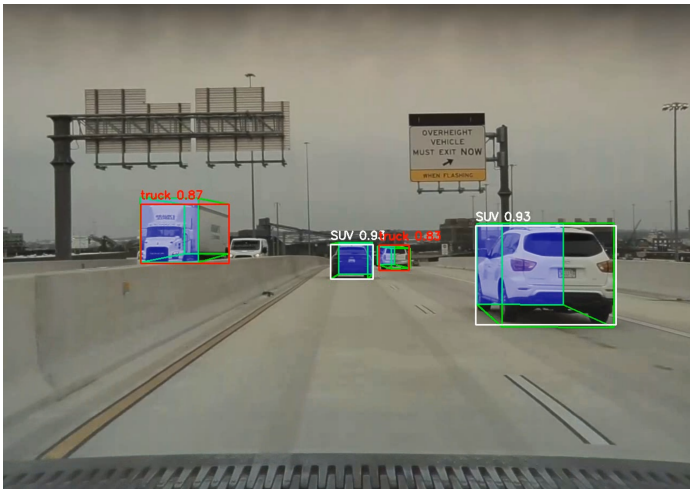

Fig. 10. Vehicle Pose Detection

were systematically saved in a JSON file for smooth interaction with Blender's rendering pipeline. Blender scripts accurately draw vehicles in the 3D simulation environment using this structured data, with realistic location and orientation. This complete technique achieved excellent realism in visualizing vehicle dynamics, which is crucial for autonomous driving simulations that rely on precise modeling of real-world traffic situations. The outputs for vehicle pose detection is displayed in Figure 10.

B. Traffic Light Arrows

In our pipeline for recognizing traffic signal arrows, we included a dedicated module that uses traditional image processing techniques to rigorously assess a region-of-interest (ROI) for directional indicators. The technique starts with converting the candidate ROI to grayscale and applying histogram equalization to improve contrast, followed by adaptive thresholding to create a binary image. Contours are then retrieved from this binary image; if no contours or insufficient vertex details are found, the algorithm marks the direction as "unknown." For robust detection, the largest contour is chosen and approximated to a polygon using arc length-based

Fig. 11. Traffic Light Detection with Arrows

algorithms, guaranteeing that only contours with a significant number of vertices (indicating an arrow form) are analyzed.

Once a plausible contour has been identified, we calculate the smallest area rectangle enclosing it and determine its orientation angle. If the measured angle is beyond the predicted range (less than -45°), it is normalized by adding 90°. The normalized angle is then interpreted using heuristic thresholds: angles around 0° suggest a rightward-pointing arrow, angles more than 75° are classified as left-pointing, and intermediate angles are presumed to correlate to an upward orientation. This module detects the directionality of arrow signals within traffic lights using contour analysis and geometric heuristics. The generated directional labels are then saved in JSON format alongside other detection details for correct rendering and subsequent processing in Blender for self-driving applications. The output for traffic light detection with arrows is shown in Figure 11.

C. Pedestrian Pose Detection

For pedestrian pose identification, we used the OSX model [9], a cutting-edge framework built for precise human position estimate. The model recognizes pedestrians in the scene and creates detailed 3D posture information, which is exported in the form of OBJ files. These OBJ files contain exact geometric representations of pedestrian poses, including as joint locations and limb orientations, and may be imported directly into Blender without any further processing. This streamlined approach ensures that recognized pedestrians are represented in the simulation environment in realistic and anatomically precise positions, hence improving the visual quality and contextual realism required for autonomous driving scenarios. The output for Pedestrian Pose detection is displayed in Figure 12.

D. Traffic Signs

In our traffic sign detection pipeline, we used the YOLOWorld model [6] and EasyOCR to detect stop and speed

Fig. 12. Pedestrian Pose Detection

Fig. 13. Stop Sign Detection

limit signs in video frames. The YOLOWorld model was customized with specific labels for various speed restrictions and stop signs, which were applied to each video frame to provide detection outputs. The model returns bounding boxes for each frame, along with their corresponding confidence scores. When a road sign is recognized, the matching bounding box is used to crop the region of interest, which is then processed by EasyOCR to extract textual content. This method entails analyzing OCR data to isolate numerical values and validating them as appropriate speed limits based on expected criteria (for example, ensuring that detected numbers are multiples of five).

Each frame's detected metadata is assembled into a structured JSON file, which includes bounding box coordinates, confidence levels, and, if applicable, extracted speed limit values. This organized output enables for seamless integration with Blender's rendering workflows, guaranteeing that each traffic sign is correctly positioned and represented in visualization stages. This complete methodology successfully integrates object identification, optical character recognition, and video processing to produce high-fidelity traffic sign detection in self-driving simulations.

The stop sign, speed limit detections are shown in Figures 13, 14.

E. Road Signs

In our pipeline, for identifying road signs on the ground, such as arrows, we use the output from the same Mask R-CNN model[8] used for lane detection. The model gives bounding boxes around the road signs along with a mask of the road sign. This mask is then processed to find which type of road sign it is. Using the mask, we first extract the largest contour from the mask. Then using Principal Component Analysis on the contour points to find the primary direction through eigenvectors. Using these eigenvectors, we calculate the angle which are then thresholded using trial and error to classify

Fig. 14. Speed Limit Detection

the arrow as point 'up', 'right' or 'left'. This directional information is saved in JSON format in the same file for lane detection.

F. Miscellaneous Object Detection

We used the Detic framework to detect objects necessary for contextual road scene analysis, such as trashcans, traffic cones, and traffic cylinders. Detic's robust architecture allows successful detection across a wide range of environmental conditions by generating precise bounding box coordinates, confidence ratings, and object labels for these things. Detic's detection output was stored in a structured JSON file, which acted as an intermediary data format for smooth integration with Blender's rendering pipeline. The micsellaneous objects detected are shown in figure 15, 16 and 17.

Fig. 15. Trash Can Detection

Fig. 16. traffic Cone Detection

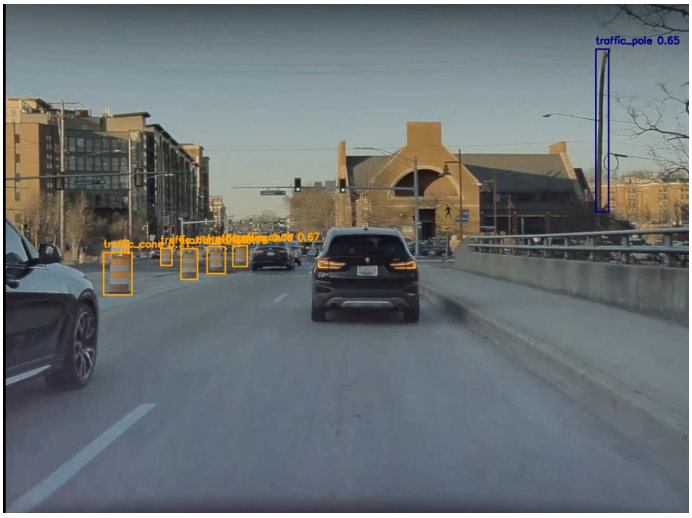

Fig. 17. Traffic Cylinder Detection

IV. Phase 3 : Phase 3: Bells and Whistles

A. Vehicle Brake Lights and Indicator Detection

We used a region-based intensity thresholding technique to detect and classify vehicular signaling lights within traffic scenes. We extracted predefined regions of interest (ROIs) from vehicle bounding boxes using normalized coordinates, focusing on four critical areas: left tail (0.00-0.18, 0.30-0.55), right tail (0.82-1.00, 0.30-0.55), center windshield (0.35-0.65, 0.40-0.60), and top center (0.35-0.65, 0.05-0.25). We used grayscale conversion followed by binary thresholding at an intensity value of 200 to each extracted region, allowing us to isolate high-luminance pixels typical of active signaling lights. We quantified illumination present by calculating an active pixel ratio (bright pixels/total pixels), with a 0.03 threshold set to reduce false positives while retaining detection sensitivity.

We classified vehicle signaling intentions using conditional logic based on the geographical distribution of measured brightness patches. We distinguished three signaling states: brake application (defined as simultaneous activation of more that 2 light regions), left-turn indication (isolated activation of the left tail region), and right-turn indication (isolated activation of the right tail region). To provide a probabilistic estimate of detection reliability, we calculated a confidence metric using the active pixel ratio, scaled it by a factor of ten, and capped it at 0.8. All the information was then stored in a JSON file to facilitate proper rendering.

B. Parked and Moving Vehicle Identification

In our system for detecting parked from moving automobiles, we used RAFT [10]—a cutting-edge optical flow model—to generate dense motion fields between successive frames. To maintain constant proportions, we integrated the RAFT architecture into our pipeline by first importing and prepping sequential image frames with a specialized image loading function and an input padding tool. After being suitably pre-processed, the images were routed via the RAFT network, which iteratively refines the predicted flow until a high-resolution flow map is produced. This flow map captures the pixel-wise displacement between subsequent frames with low noise and high precision, allowing us to successfully capture motion characteristics indicating vehicle movement.

The computed optical flow outputs are then converted into an understandable display by mapping the flow vectors to RGB pictures, with the flow magnitude and direction clearly indicated. Vehicles with significant pixel displacement between frames are classed as moving, whereas those with limited or no flow are identified as parked. The resulting flow visualizations not only provide qualitative insight into the scene's dynamics, but they also act as strong quantitative indicators when paired with further vehicle detection data.

Despite the RAFT model's sophisticated capabilities, we faced major difficulty in accurately determining vehicle motion states. The main challenge came from our own vehicle's motion (ego-motion), which injected relative motion into all parts of the scene. This resulted in a basic ambiguity: even

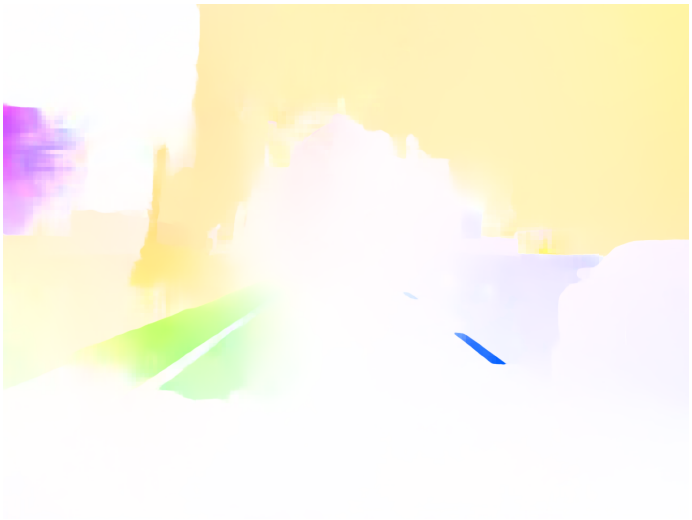

Fig. 18. Optical Flow

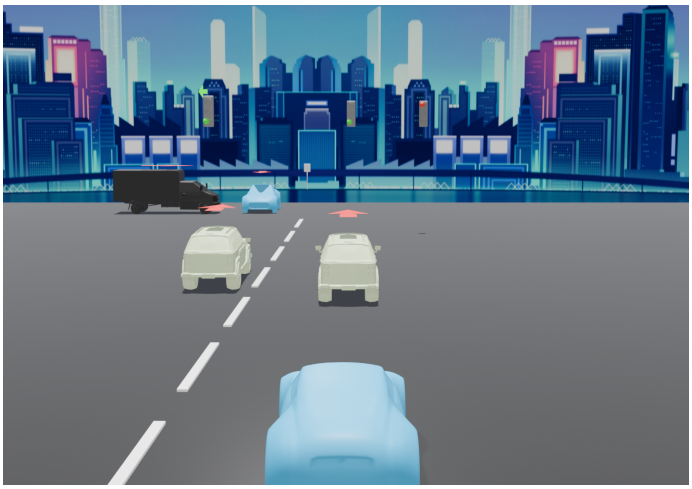

Fig. 19. Final Output 1

motionless vehicles exhibit visual flow when caught by a moving camera. This ego-motion effect hindered our attempts to improve the optical flow outputs for correct motion information extraction.

Furthermore, we discovered that mere thresholding of flow magnitudes was insufficient for accurate classification. When our vehicle drove quicker than the car ahead of us, the relative motion may fall below our detection threshold, allowing the system to falsely categorize plainly moving vehicles as stationary. Conversely, legitimately parked automobiles would report high optical flow merely due to our camera's movement, resulting in false positives.

The optical flow maps are displayed in Figure 18

Final Outputs are displayed below:

Fig. 20. Final Output 2

Fig. 21. Final Output 3

Fig. 22. Final Output 4

Fig. 23. Final Output 5

Fig. 24. Final Output 6

V. Extra Credit

A. Speed Bump Detection

We extended our detection system to include a module for speed bump sign recognition using optical character recognition (OCR). We added this to our original part of pipeline designed to identify stop signs and speed limits, but it was later upgraded to analyze and interpret additional speed bump-related signs. When the program detects a speed bump sign, it adds the corresponding metadata to a JSON file for subsequent study. Based on US road design principles, which typically require a 100-foot spacing between the speed bump sign and the real speed bump, we placed a speed bump marker 100 feet beyond the detected sign in blender. Our rendered output with speed bump is displayed in Figure 25.

VI. Challenges Faced

Several technological challenges arose during the implementation of our full vehicle identification and scene interpretation system, presenting prospects for future study. Our approach to vehicle categorization used an ensemble methodology that combined several cutting-edge detection frameworks, including Detic, YOLO World, and YOLO 3D, with proprietary taxonomies. Despite this multi-model integration technique, fine-grained vehicle categorization (differentiating between sedans, SUVs, hatchbacks, buses, trucks, pickup trucks, vans, motorcycles, and bicycles) remained difficult. The integration of several detection systems, while modestly improving classification accuracy, considerably increased computing overhead and inference latency, emphasizing the need for more efficient classification architectures suited specifically for the vehicle classes.

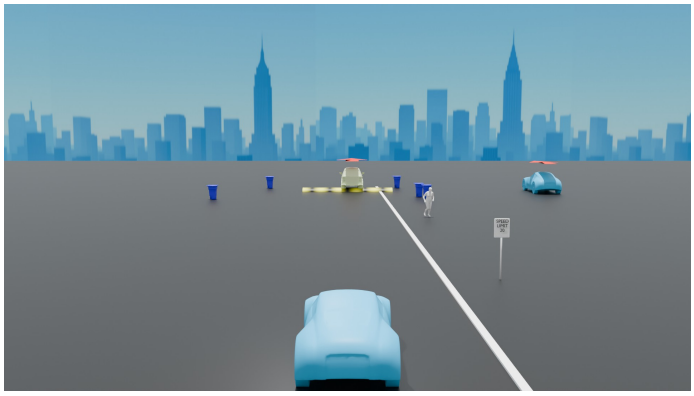

Fig. 25. Final Output 6

Furthermore, the spatial localization accuracy of vehicle instances raised new concerns. YOLO 3D's bounding box predictions occasionally contained errors that propagated to subsequent processing steps, particularly orientation estimation. These geometric defects manifested as misalignments between the observed vehicle posture and its real orientation in three-dimensional space. Such anomalies underscore the need for more robust geometric reasoning systems, which may integrate contextual information from road layout and trajectory limitations, to adjust orientation calculations when detection limits are incorrect. We wanted to train the model more to improve its effectiveness and fix the geometric inconsistencies, but such refinement was not possible given the project's time restrictions.

The classical computer vision algorithms used for traffic signal interpretation showed low durability under changing environmental conditions. The recognition of traffic light colors and directed arrows was especially sensitive to lighting changes, partial occlusions, and perspective distortions. These constraints suggest that a hybrid method combining classical image processing with deep learning algorithms trained on varied traffic signal datasets could produce more consistent results in a variety of weather and illumination conditions.

Perhaps the most major technological challenge encountered was accurately estimating and applying depth information during the Blender rendering phase. Despite using modern monocular depth estimation techniques, achieving consistent depth scaling across whole video sequences proved exceedingly difficult. Our research indicated that absolute depth estimate necessitates at least two reference points with known

distances to establish correct scaling factors. The relative depth scaling method used produced acceptable results for individual frames but failed to preserve consistent spatial relationships throughout extended sequences. This constraint highlights an important research direction: the creation of temporal consistency frameworks for monocular depth estimation that can use dynamic scene comprehension to maintain coherent depth mapping across multiple video frames.

These difficulties emphasize the cutting edge of computer vision research in autonomous driving and scene interpretation applications. They underline the importance of interdisciplinary techniques that integrate developments in deep learning architectures with domain expertise in vehicular dynamics, traffic systems, and 3D scene reconstruction. Future research directions could include the integration of transformer-based architectures for finer-grained classification, multi-frame consistency frameworks for robust orientation estimation, adaptive filtering techniques for traffic signal detection, and self-supervised learning approaches for temporally consistent depth estimation in dynamic driving scenarios.

REFERENCES

[1] Ultralytics. (2024). *YOLOv11: A state-of-the-art object detection model*. GitHub repository. Available at: https://docs.ultralytics.com/models/yolo11/

[2] X. Zhou, V. Koltun, & P. Krähenbühl. (2022). *Detic: Detecting Twenty-thousand Classes Using Image-level Supervision*. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.* (pp. 10883-10893). Available at: https://github.com/facebookresearch/Detic

[3] Ultralytics. (2022). *YOLOv5: State-of-the-art object detection at the speed required for real-time processing*. GitHub repository. Available at: https://github.com/ultralytics/yolov5

[4] Debugger Cafe. (2023). *Lane Detection using Mask RCNN*. Available at: https://debuggercafe.com/lane-detection-using-mask-rcnn/

[5] Y. Li et al. (2024). *Depth Anything v2: Stronger Rural and Urban Monocular Depth Estimation*. arXiv preprint arXiv:2404.04048. Available at: https://github.com/LiheYoung/Depth-Anything

[6] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan. (2024). *YOLO-World: Real-Time Open-Vocabulary Object Detection*. arXiv preprint arXiv:2401.17270, 2024. Available at: https://docs.ultralytics.com/models/yolo-world/

[7] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. (2016). *3D Bounding Box Estimation Using Deep Learning and Geometry*. arXiv preprint arXiv:1612.00496, Dec. 2016. Available at: https://github.com/ruhyadi/YOLO3D?tab=readme-ov-file

[8] W. Abdulla. (2017). *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. GitHub repository. Available at: https://github.com/matterport/Mask_RCNN

[9] J. Lin, A. Zeng, H. Wang, L. Zhang, and Y. Li. (2023). *One-Stage 3D Whole-Body Mesh Recovery with Component Aware Transformer*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 21159-21168, 2023. Available at: https://github.com/IDEA-Research/OSX

[10] Teed, Z., & Deng, J. (2020). *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow*. Proceedings of the European Conference on Computer Vision (ECCV), pp. 402–419. Available at: https://github.com/princeton-vl/RAFT