RBE 549 P3 : Einstein Vision Using 6 Late Days

Farhan Seliya
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: faseliya@wpi.edu

Sai Hitesh Viswasam
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: sviswasam@wpi.edu

***Abstract*—This report presents our project on developing a user-friendly and visually intuitive dashboard for a multi-functional Advanced Driver Assistance System (ADAS) for autonomous vehicles. This work presents a comprehensive perception pipeline that combines classical computer vision techniques with existing deep learning models to construct a 3D Digital twin of the surrounding environment for better visualization and autonomy. The pipeline captures and visualizes both basic and advanced scene features such as detecting lanes, the pose and motion of cars and pedestrians, traffic signs and lights, traffic scene objects, speed bumps, brake lights and indicators.**

I. Introduction

A major challenge in developing an Advanced Driver Assistance System (ADAS) is the lack of intuitive visualizations that help users understand what an autonomous system perceives and how it reasons. To address this, we develop a visualization system inspired by Tesla's latest dashboard, aiming to bridge the gap between perception and human interpretability. Using real-world driving videos from a 2023 Tesla Model S, our pipeline combines classical computer vision and deep learning models to detect key scene elements such as lanes, vehicles, pedestrians, and traffic signals. The result is a rendered 3D digital twin that provides both functional insights for debugging and a visually rich interface for user trust and engagement.

Using our proposed pipeline, we ran inference on one frame per second from the front camera view of Tesla videos provided, detecting object types and their 3D positions and poses. The results were stored in `.csv` files which were taken as input in Blender. Using these files, we rendered accurate, scene-specific 3D visualizations using both provided and custom object models.

II. Phase I : Basic Features

IN this phase, we developed our pipeline to detect basic features in a driving scene such as lanes, vehicles, pedestrians, traffic lights and stop signs.

A. Lane Detection

To identify and detect the lanes in the scene, we used a pre-trained Mask RCNN model [1] trained on a road scenes dataset that can identify different instances of lanes such as a solid line, dotted line, divider line, etc. We stored the information of the pixel co-ordinates of the masks obtained for projecting them into 3D later.The detection results are shown in fig.1.

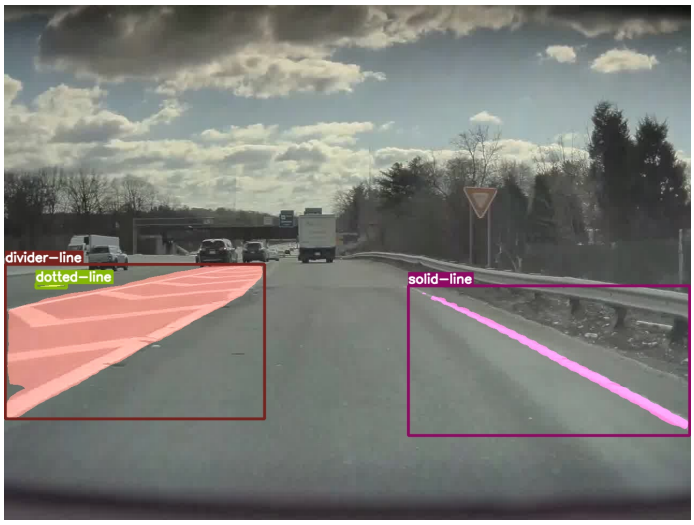

Fig. 1: Segmentation of different lane instances using the Mask RCNN model

For identifying the colors of the lanes, we employed a classical computer vision approach of color thresholding over the bounding boxes obtained from the network. We initially used thresholding in the HSV color space to identify and segment yellow color from white, the results of which turned to out to be very less accurate, due to the different lighting conditions across the scenes. Eventually, we employed segmentation in the HLS color space instead, which we found to be robust to detecting yellow across multiple lighting conditions. The result of HLS segmentation for yellow lanes can be seen in fig.2 and 3.

Once the coordinates of the lanes were stored and the color of each lane was segmented accordingly, we had to project them into Blender, using the Pinhole projection model. We followed the following steps to project the image points into 3D.

1) Extract the coordinates of each mask obtained from the

Fig. 2: Thresholding of Yellow lanes using the HLS color space.

Lane detector network.

2) Sample points along the mask coordinates to fit a second order polynomial to approximate the curves.
3) Plot the sampled points into Blender using the pinhole projection equation given by:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R|T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Here we assumed that R is identity and T is a zero vector, since car's frame is considered as the world frame itself. Since we know that lanes on the road lie on the ground, the 3D points are assumed to be planar and lie at ground level with zero height (Z=0).

4) Once the points are projected in Blender, we use interpolation within Blender to join the points to form a plane to plot inside Blender.

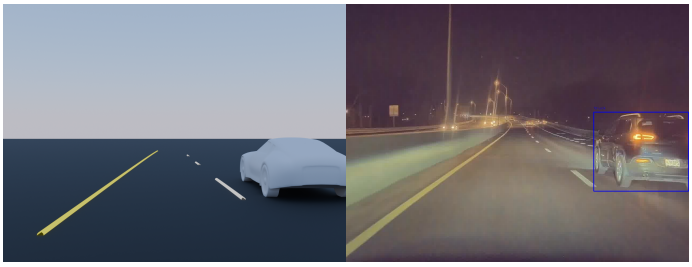

Fig. 3: Projection of lanes into Blender along with the color information.

B. Vehicle, Pedestrian and Stop Sign Detection

To develop an effective object detection model capable of identifying vehicles, pedestrians, and street signs, we selected YOLOv9 [2] as the most suitable framework. Leveraging its pretrained capabilities, the model was employed to detect five specific vehicle subcategories: Car, Bus, Truck, Bicycle, and Motorcycle. Similarly, the same model was utilized to detect pedestrians, traffic lights, and road signs, facilitating broader traffic scene understanding. For each identified object, the model outputs a bounding box, segmentation mask, and a corresponding confidence score. Figure 4 highlights an example of the YOLOv9 model in action.

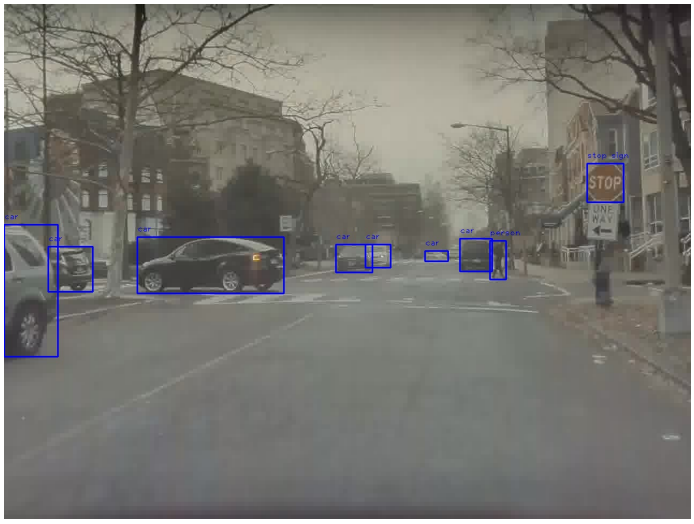

Fig. 4: Detection of Vehicles, Pedestrian and Stop Sign using YOLOv9

To enable 3D scene visualization in Blender and support 2D-to-3D projection, our approach leverages the Pinhole projection model, defined by the equations:

$$x = \frac{(u - c_x) \cdot z}{f_x}, \quad y = \frac{(v - c_y) \cdot z}{f_y}$$

where f_x and f_y represent the focal lengths, and c_x and c_y denote the principal point coordinates. Here, u and v are the pixel coordinates of the object in the image frame, and z is the depth value, obtained from a depth network. We used the Marigold Depth Estimator network [3] to estimate the relative depth between the objects in the scene. The depth map obtained from Marigold is shown in fig.5.

The resulting x and y correspond to the coordinates in the 3D world frame. Since the visualization is built for a specific camera, the intrinsic parameters are known beforehand, allowing accurate projection from image coordinates to 3D space.

As an initial estimate for the object's pixel coordinates, we used the centroid of its bounding box 6. The object's depth was obtained using the previously described depth estimation network. Leveraging the known camera intrinsics, pixel coordinates, and metric depth, we computed the object's 3D position and visualized it in Blender. A rotation matrix was applied to transform the 3D points relative to the camera frame. Figures 6 and 7 highlight a sample scene of the implemented method and its corresponding blender visualization.

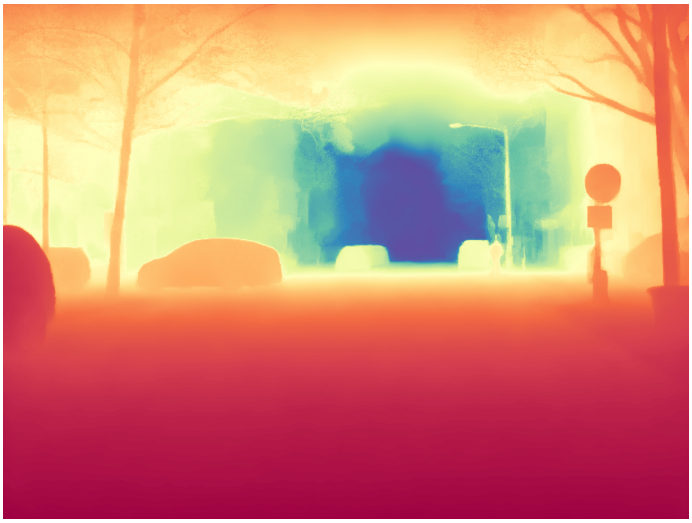

Fig. 5: Estimating relative depth using the Marigold depth estimator network

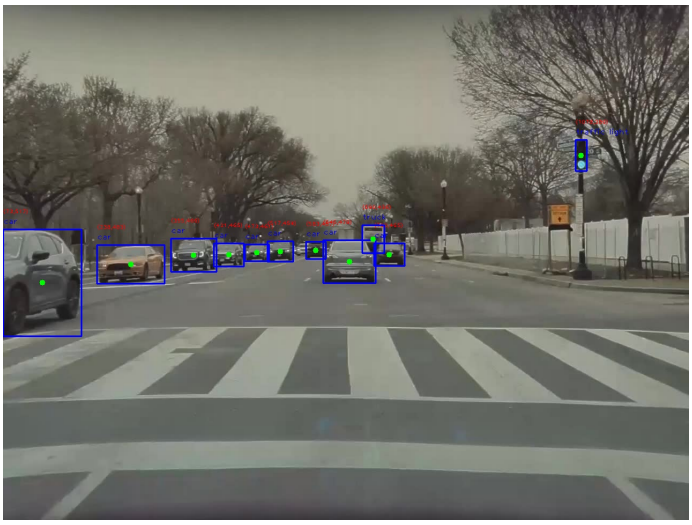

Fig. 6: Estimating the Object's position in the image frame through the coordinates of the centroid of the bounding box

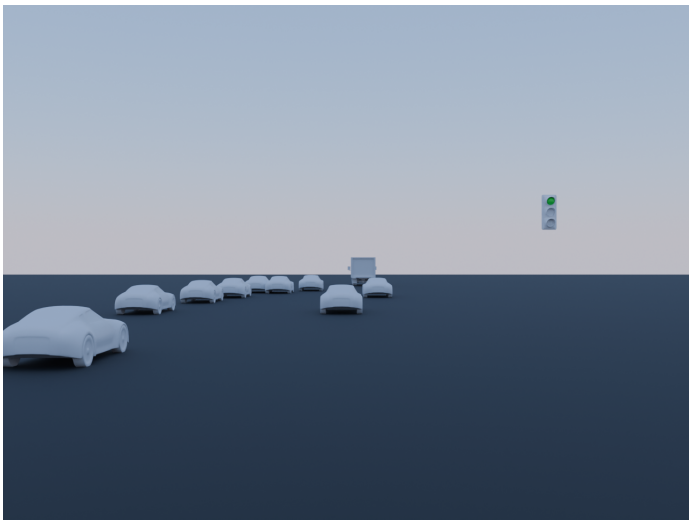

Fig. 7: Rendering the objects in 3D using the position obtained from the centroid of the bounding box and the Pinhole Projection Equations

C. Traffic Light Detection and Classification

Using the YOLOv9 object detection model, we successfully detected the presence of traffic lights; however, the model was not able to identify their color state.To address this limitation, we initially employed a pretrained model based on the YOLOv3 framework. Trained on six distinct classes, this model was capable of classifying traffic light signals—such as green as "go," red as "stop," yellow as "warning," and directional indicators like the left arrow as "goLeft," as shown in figure 8.

Fig. 8: Incorrect Classification and limited Robustness of YOLOv3 network in identifying traffic signals

1) Challenges Faced: However, this approach proved ineffective in delivering consistent results across sequences of frames, rendering it unreliable for deployment. Additionally, its performance significantly declined when traffic lights were distant or not clearly visible, limiting its robustness in real-world scenarios, as shown in figure 8.

2) Solution Approach: To enhance traffic light color classification, we adopted a classical approach leveraging a color-based heuristic for state recognition. YOLOv9 was used to detect traffic lights and extract regions of interest (ROIs) from the image. Each ROI was first analyzed to determine if it depicted the front or back side of the traffic light using saturation and brightness thresholds in the HSV color space as in fig.10. For ROIs likely showing the front, the region was divided into three vertical segments corresponding to red, yellow, and green positions. The segment with the highest brightness was used to infer the active signal color—red, yellow, or green—based on its relative position. The correct classification outputs of traffic lights using this method are shown in fig.9 and 11

Fig. 9: Corrected Classification of Traffic Lights and Signals using HSV Thresholding

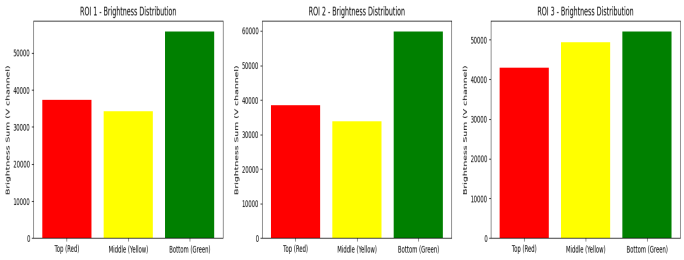

Fig. 10: Identifying color of the traffic light based on brightness values of colors obtained from HSV Thresholding

D. Issues Encountered & Implemented Solution

Testing our centroid-based object localization revealed inaccuracies, especially with occluded objects, due to unknown or varying object heights affecting 3D projections. To improve reliability, we instead used the midpoint of the bounding box's bottom edge and projected it onto the ground 12. For street signs, we assumed a fixed mounting height and adjusted the projection equation accordingly for better spatial accuracy.

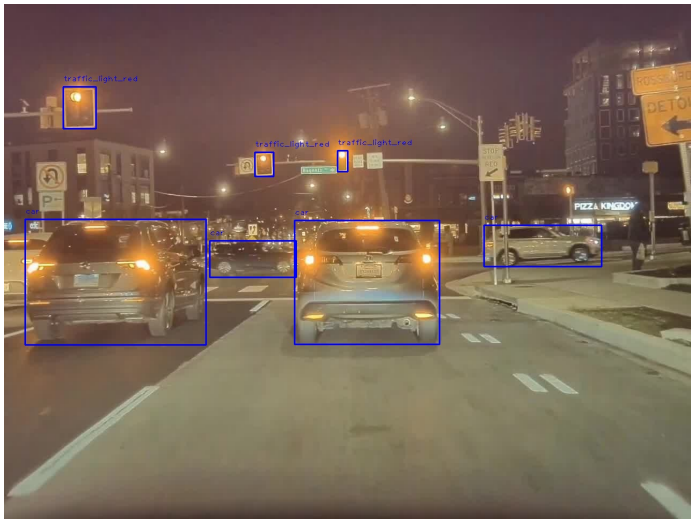

Fig. 11: Classification of traffic lights using HSV Thresholding

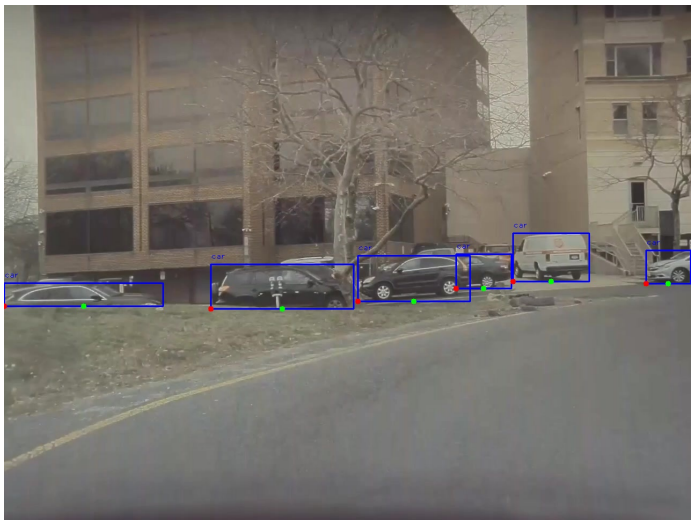

Fig. 12: Estimating objects position in the image frame using midpoints of the lower edge of the bounding box

III. Phase II : Advanced Features

In this phase, we extend the previous work by refining the vision system by incorporating enhanced functionality and additional features to provide greater granularity, enabling more informed decision-making.

A. Vehicle Classification & Subclassification

Since YOLOv9 is limited to a predefined set of classes, vehicle *sub-classification* posed a challenge. To address this, we explored **Facebook's DETIC** [4], a promising *zero-shot detection* framework capable of generalizing to unseen categories, as in 16, 17 and 18. DETIC extends the COCO and LVIS class sets and leverages **CLIP**, a vision-language model, to support open-vocabulary detection. This allows us to define custom class names (e.g., sedan, SUV, pickup truck) as textual inputs, enabling fine-grained vehicle classification.

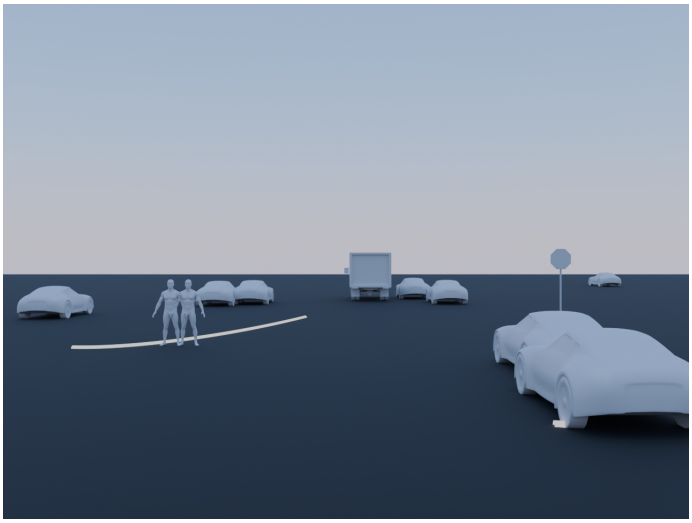

Fig. 13: Basic 3D Rendering of a scene with features such as cars, pedestrians and stop signs

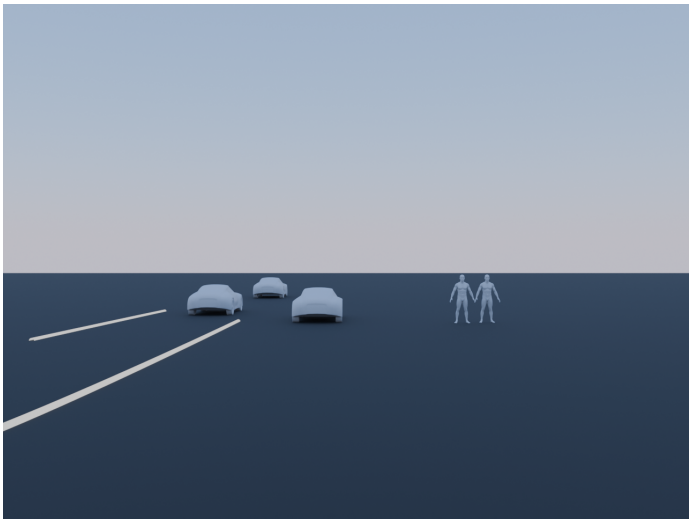

Fig. 14: Basic 3D Rendering of a scene with cars and pedestrians

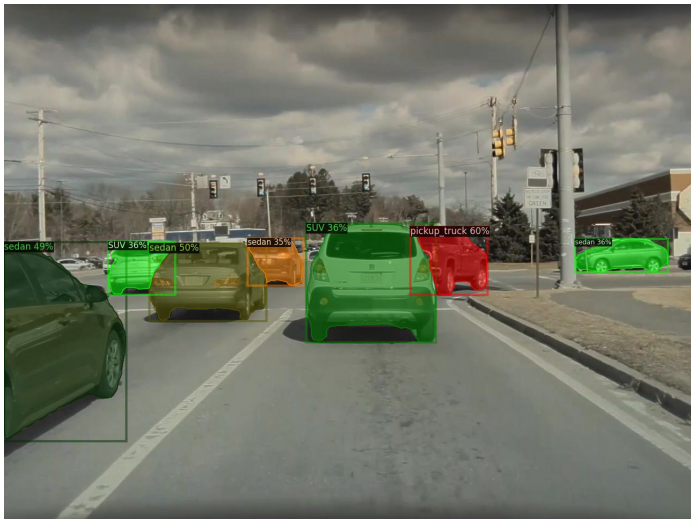

Fig. 15: Detection of sub classes of Cars using Facebook DETIC

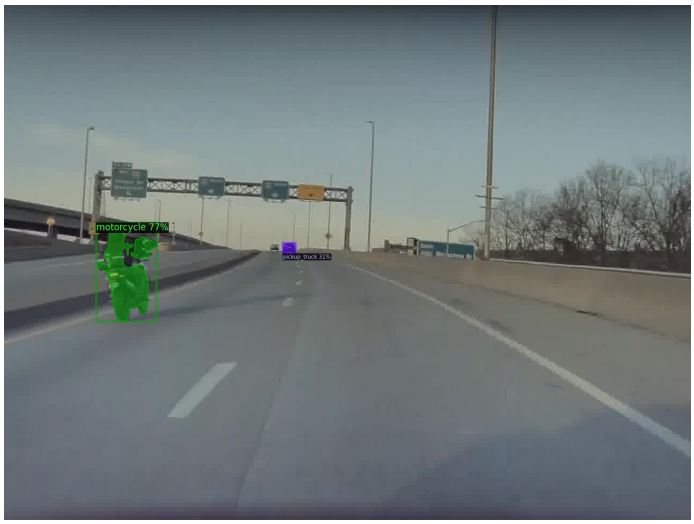

Fig. 16: Detection of Motorcycle using Facebook DETIC

In this phase, we replaced YOLOv9 with DETIC to improve classification granularity. As shown in Figure15, DETIC effectively identified car sub-types. Additionally, it supports confidence threshold tuning to minimize false positives, which improved robustness across varied scenes. However, the approach has some limitations: predictions tend to have lower confidence scores due to vocabulary generalization, and the inference speed is significantly slower compared to YOLO-based methods.

B. Vehicle Pose Estimation

We estimate vehicle orientation using a pretrained **YOLO3D** [5] model with a ResNet backbone, trained on the KITTI dataset. The 2D bounding boxes detected by DETIC are input to the YOLO3D regressor, which outputs the corresponding 3D bounding boxes. From these, we extract the yaw angle to accurately orient vehicles within the Blender environment. Vehicle positions are computed using the projection approach detailed in the previous section.

The estimated orientations using YOLO3D were generally accurate across various vehicle subtypes. However, we observed unreliable yaw predictions in cases of occlusions or when vehicles were positioned very close to the camera. To address such edge cases, a more robust alternative is **Omni3D** [6] by Facebook Research, which is optimized for accurate vehicle pose estimation under challenging conditions. Due to time constraints, we were unable to integrate this model, but it presents a promising direction for future improvements.

Figure 19, 20, 21 illustrates a typical successful detection, with the shaded blue face indicating the vehicle's heading. In contrast, Figure 22 depicts a failure case where the predicted yaw is inaccurate for nearby vehicles.

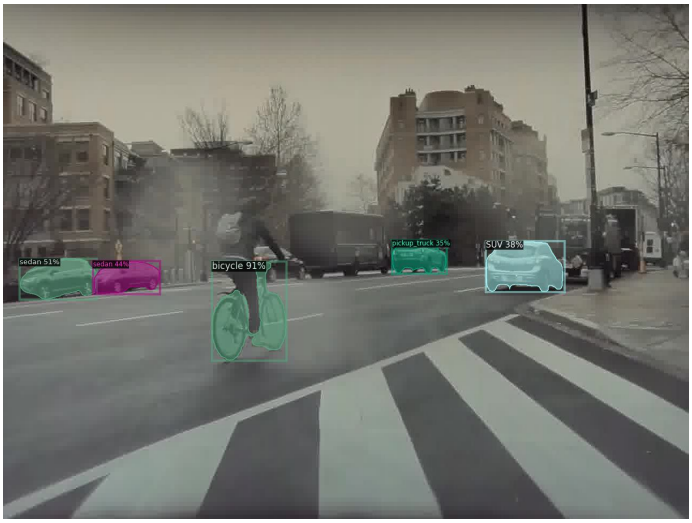

Fig. 17: Detection of Bicycle using Facebook DETIC

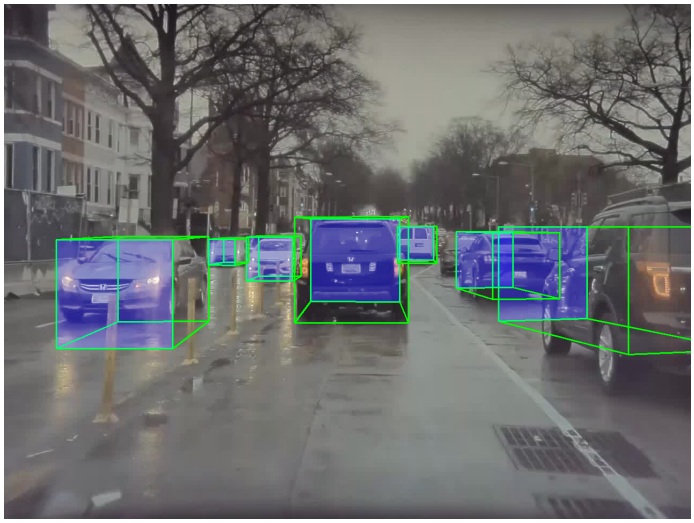

Fig. 19: Detection of vehicle poses using YOLO3D

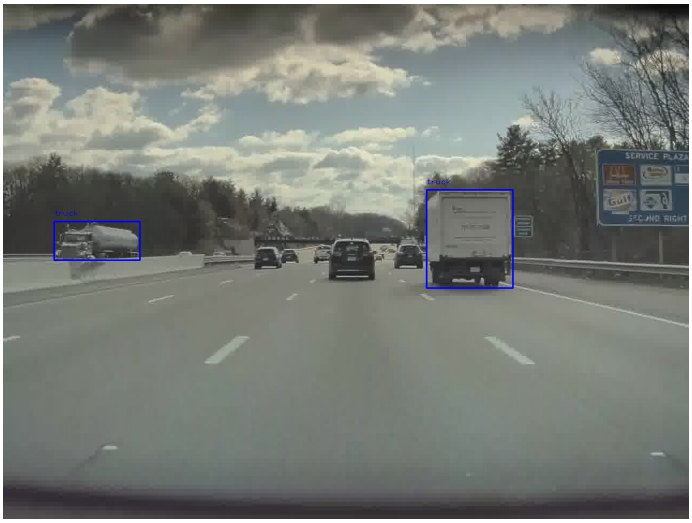

Fig. 18: Detection of Trucks using Facebook DETIC

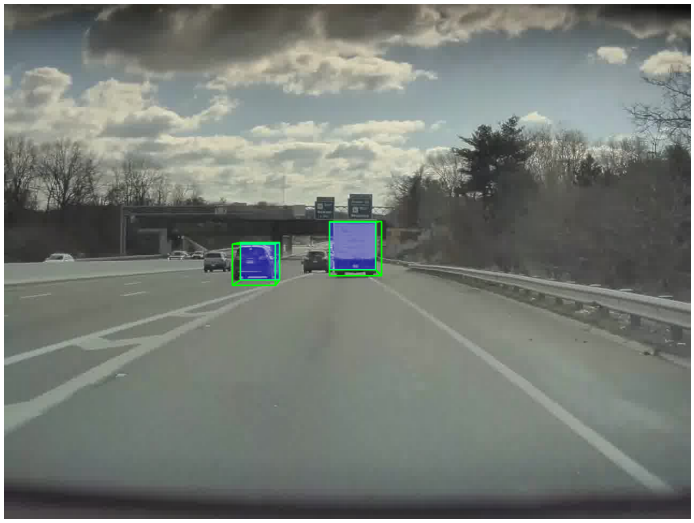

Fig. 20: Detection of vehicle poses using YOLO3D

C. Miscellaneous Objects

Beyond vehicles, our system also identifies and projects common roadside objects such as dustbins, traffic poles, cones, and traffic cylinders. **DETIC** proved effective in detecting these elements, maintaining high accuracy even under occlusion as shown in 23, 24, 25 and 26. Their positions are computed using the bottom-edge projection method, leveraging bounding boxes from DETIC and depth information from the previously discussed Depth Estimator. It is worth noting that due to the visual similarity between traffic cones and cylinders, the model occasionally misclassified them; nonetheless, the overall detection performance remained robust.

D. Detection of Road Signs

In this phase we had to detect two types of road signs - the road signs on the ground such as arrows and the speed limit signs alongside the stop signs. The detection of road signs on the ground is done by the Mask RCNN Lane detector network from the Phase 1 itself as shown in fig.27.

To detect the speed limit signs and other signs on the road, we used DETIC again with custom defined class names as mentioned previously and detected different street signs including speed limit signs, One way signs, No U-Turn signs etc as shown in 28 and 29. Additionally, after detecting the sign, we also crop the bounding box from the image as a region of interest and project the cropped image onto the empty sign asset while rendering in Blender as shown in 30.

E. Traffic Light Arrow Classification

We initially employed a pretrained model based on the YOLOv3 framework [7] to classify traffic light states into six distinct categories: *red (stop)*, *green (go)*, *warningLeft*, *goLeft*, *stopLeft*, and *warning*. While effective in many close-range

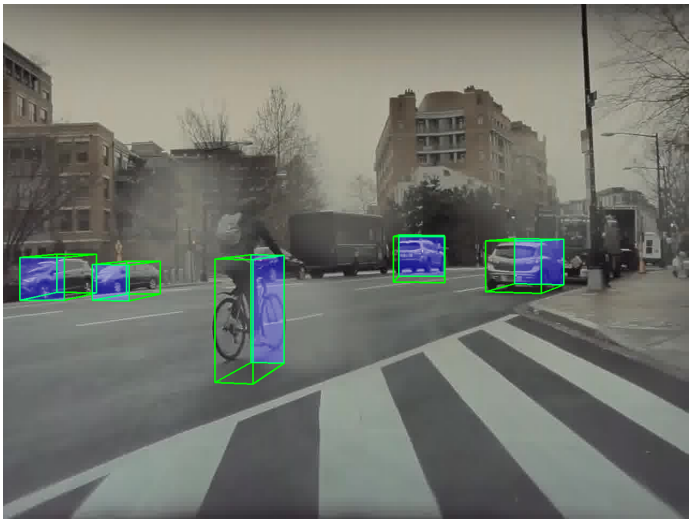

Fig. 21: Detection of vehicle poses using YOLO3D

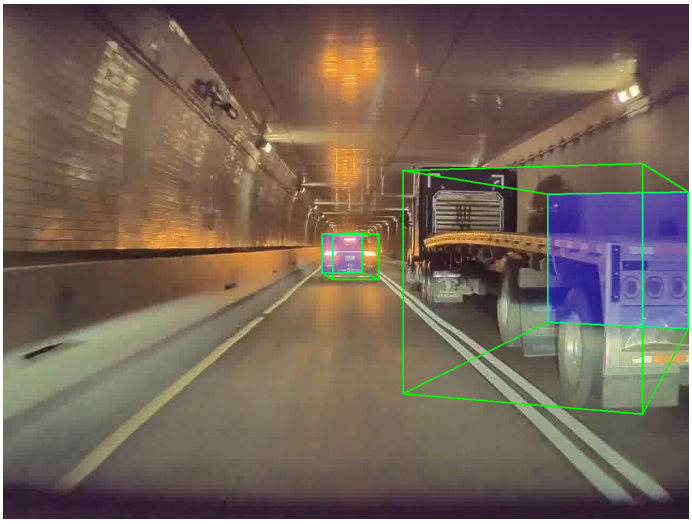

Fig. 22: Failed case of Vehicle pose estimation

Fig. 23: Detection of Trash Cans using DETIC

Fig. 24: Detection of Traffic Poles using DETIC

scenarios, this approach suffered from false positives, such as misclassifying an upward arrow as a goLeft signal, largely due to limited training classes and poor generalization to varied conditions.

To address these limitations, we extended a classical vision-based pipeline that applies adaptive color thresholding to segmented Regions of Interest (ROIs) for detecting traffic light colors. For directional inference from arrow-shaped signals, we convert the ROI to grayscale, enhance contrast via histogram equalization, and apply adaptive thresholding to isolate shapes. The largest contour is approximated, and a minimum area bounding rectangle is used to estimate the arrow's orientation. Based on the rectangle's angle, we heuristically determine the direction: angles near 0° indicate right, near 90° indicate left, and intermediate values suggest upward direction. This hybrid method proved more reliable across varied scenes and occlusions as shown in 31.

F. Pedestrian Pose Estimation

For estimating the pose of the pedestrians in the scene, we used a pretrained network OSX [8]. This network takes in the RGB images as input, detects the position and pose of the pedestrians present in the image and outputs a `.obj` file of the pedestrians directly which can be imported into Blender. Similar to projecting the other objects from above sections, we used the relative depth of the detected pedestrians and the pinhole projection equation to get the exact location of the pedestrians in 3D. The detection is shown in fig.32.

IV. Phase III : Bells and Whistles

A. Brake light and Indicators

We designed a method to identify and classify vehicle tail lights as **ON** or **OFF** by first associating each detected light with a corresponding vehicle. Using the annotated bounding

Fig. 25: Detection of barrels using DETIC

Fig. 26: Detection of Traffic Cones and Traffic Lights using DETIC

boxes for vehicles and lights, the method checks whether a light's center lies within a vehicle's bounding box and aligns horizontally within a specified threshold, thereby establishing ownership.

To robustly detect illumination, each vehicle region is converted to the **YCbCr** color space. Specifically, the **Y** (luminance) channel is extracted, which emphasizes brightness information while reducing the influence of color. This makes it more effective in identifying illuminated regions under varying lighting conditions.

An adaptive thresholding algorithm is applied on the Y channel to highlight high-intensity areas that are likely to correspond to active tail lights. For each candidate light region, the ratio of white (bright) pixels to the total number of pixels is computed. If this ratio exceeds a predefined threshold, the light is classified as **ON**; otherwise, it is marked as **OFF**. This is

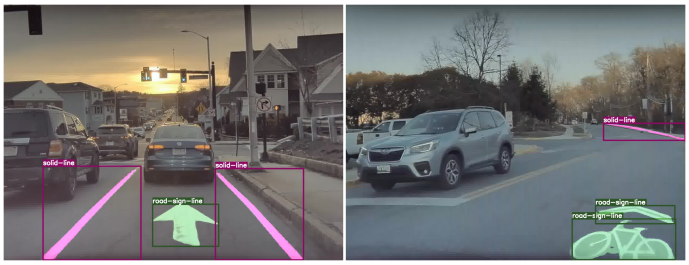

Fig. 27: Detection of Road Signs on the ground.

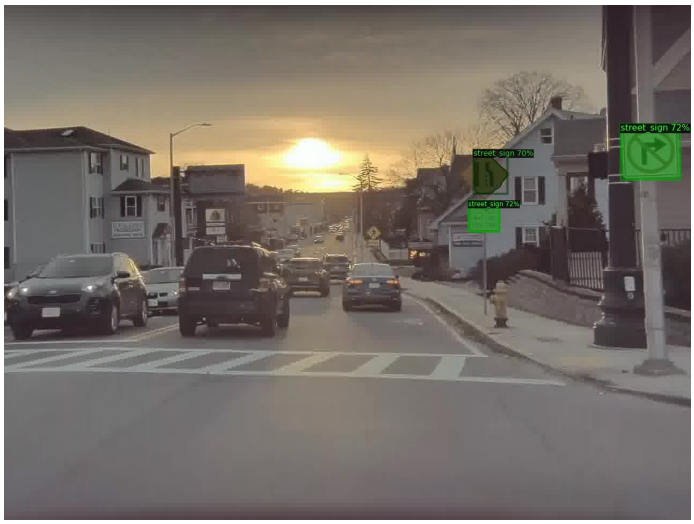

Fig. 28: Detection of Road Signs on the street

shown in 34 and 35. Additionally, an interactive graphical user interface (GUI) is incorporated to allow adjustment of thresholds and parameters for each scene. This tuning helps improve classification accuracy across different lighting environments and vehicle orientations.

The **turn signal detection system** employs a **temporal analysis approach** that tracks vehicle tail lights across sequential video frames. This method focuses on detecting the **alternating on/off brightness patterns** characteristic of turn signals by tracking the **brightness history** across frames.

- The algorithm **independently monitors** the **left and right tail lights**, analyzing their blinking patterns over time.
- It compares the blinking behavior between the left and right tail lights:
  - If only the **left** tail light blinks, it indicates a **left turn**.
  - If only the **right** tail light blinks, it indicates a **right turn**.
  - If **both lights blink simultaneously**, the system compares their brightness values:
    - If the difference in brightness between the lights is similar, it may indicate **brake lights**.
    - If the difference is not similar, it suggests an **indicator light**.

Fig. 29: Detection of Speed limit signs on the street

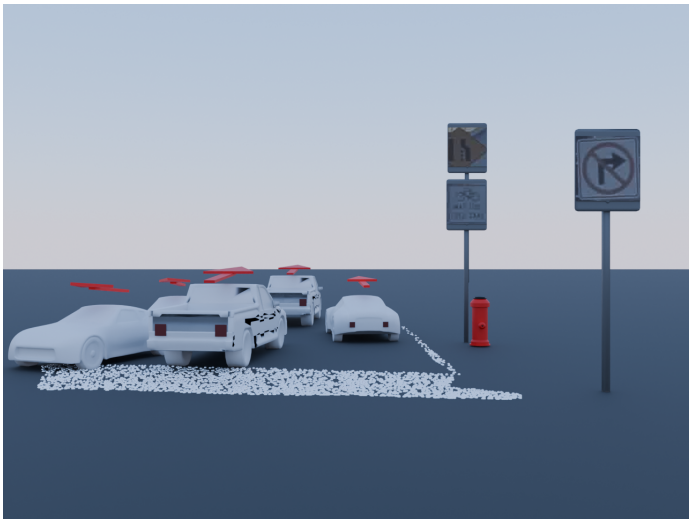

Fig. 30: Projection of detected signs in Blender

- In the case of indicator lights blinking on both sides, the system will determine the **direction of the turn** based on which side has the higher brightness.

This approach, which compares **temporal blinking patterns** between the left and right lights instead of relying on single-frame brightness values, provides a **more robust detection system**. By leveraging the **inherent temporal nature** of blinking indicators and the **relative behavior** between paired lights, the system improves turn signal detection accuracy. This is shown in 36 and its Blender rendering in 37.

B. Parked and moving vehicles

To classify vehicles as **moving** or **parked**, we analyze optical flow between consecutive frames while compensating for the camera's ego motion. Optical flow vectors

$$\mathbf{F}(x, y) = (u, v)$$

Fig. 31: Correct classification of Traffic arrows and lights using contours

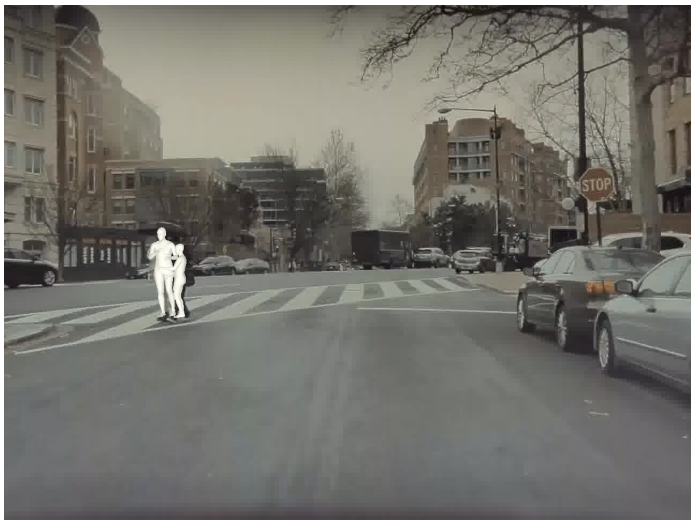

Fig. 32: Estimation of pose of the pedestrians by OSX

are computed using the RAFT model [9], capturing pixel-wise motion. Higher flow rates are represented by intensified colors in the flow images.

Initially, we estimated ego motion by analyzing the relative flow of a known static reference object (e.g., a trash can) across frames as seen in figure . By comparing flow within its bounding box, we inferred the vehicle's motion direction. However, this approach was limited by the need for manual object selection, dependency on consistent object presence across scenes, and sensitivity to video frame rate, making it unsuitable for scalable applications.

Further, we used the Sampson distance to mask moving vehicles based on their higher flow rates. However, this approach misclassified slower-moving vehicles ahead of our car as parked, since their relative motion appeared minimal, leading to low flow values indistinguishable from static objects. To

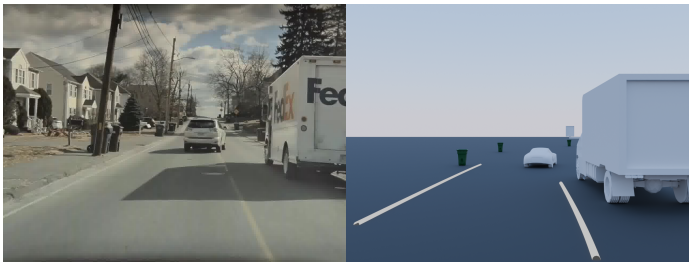

Fig. 33: Comparison of Original Image with Vehicles and Trash Can and its Blender projection.

Fig. 34: Detection of Break lights as ON using thresholding in YCbCr color space.

tackle these issues, we devised the following strategy:

1) We applied **spatial weighting** to the RAFT optical flow field, prioritizing vectors near the image center and with higher magnitude to enhance motion relevance and suppress peripheral noise.

2) To isolate true vehicle motion, ego motion is estimated by applying **DBSCAN clustering** to valid flow vectors:

$$\mathbf{V}_{\text{valid}} = \{(u_i, v_i) \mid \|(u_i, v_i)\| > \epsilon\}$$

Then, the dominant cluster's mean vector $\mathbf{E} = (u_e, v_e)$ is computed.

3) For each vehicle bounding box, the average flow

$$\mathbf{V}_B = (u_B, v_B)$$

inside the box is computed, and its relative motion to ego is

$$\mathbf{V}_{\text{rel}} = (u_r, v_r) = (u_B + u_e, v_B + v_e)$$

4) The magnitude

$$\Delta = u_r^2 + v_r^2$$

is used to classify the vehicle as moving if

$$\Delta > T$$

Fig. 35: Detection of Break lights as OFF using thresholding in YCbCr color space.

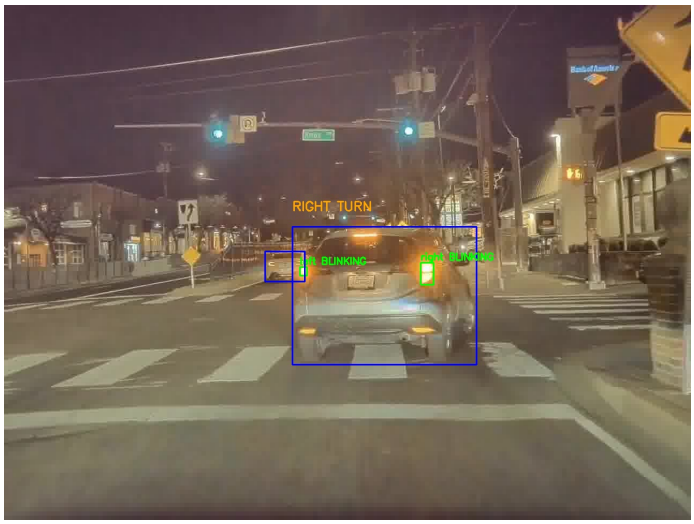

Fig. 36: Identification of RIGHT TURN based on intensity difference in the left and right break lights

(where T is a threshold, here 1.0), else it is classified as parked.

5) The direction of movement is derived using

$$\theta = \tan^{-1}\left(-\frac{v_r}{u_r}\right)$$

As compared to parked vehicles, the moving vehicles in Blender are indicated with an arrow on top of the vehicle asset, representing the estimated direction of motion. The results of motion detection using optical flow are shown in fig. 38, 39, 40, 41 and 42.

V. Extra Credit - Detecting Speed Bumps

To detect speed bumps, we leveraged the presence of nearby speed bump warning signs. Using our earlier approach for

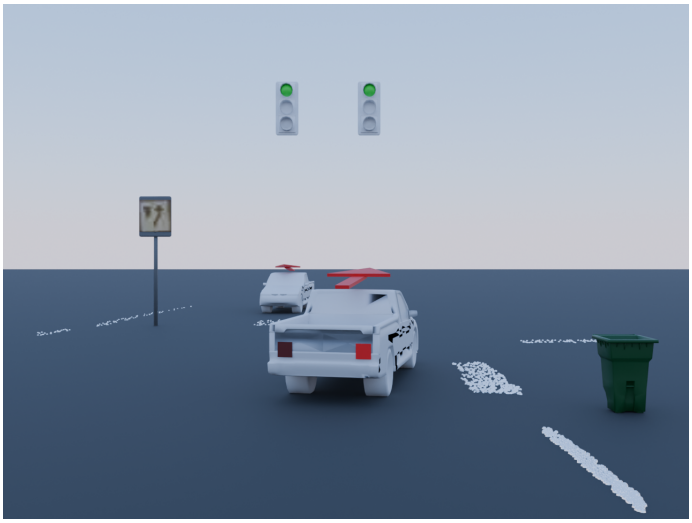

Fig. 37: Blender Rendering of the scene when the car is taking a turn

road sign detection, we first cropped out detected road signs from the images. To determine whether a given sign indicates a speed bump, we applied Optical Character Recognition (OCR) to extract text from the cropped sign image, as shown in fig.43 and 44. The extracted text was then matched against keywords such as ”hump” or ”bump”. A successful match implied the presence of a speed bump near the sign, prompting us to spawn the corresponding 3D model in Blender at the estimated location.

This method proved reliable in scenarios where a speed bump sign was placed adjacent to the bump. However, it failed in cases where no such sign was present. Alternative strategies were explored, such as identifying depth variations on the road surface or recognizing visual patterns of speed bumps directly from the image. These approaches were unsuccessful due to inconsistent depth data and high variability in the appearance of speed bumps across different scenes. The final output after rendering a speed bump having detected the sign is shown in fig.45.

Fig. 38: Identification of Parked Vehicles and direction of motion for moving vehicles using Optical Flow

REFERENCES

[1] Rath, S. R. (2023, August 14). Lane Detection using Mask RCNN – An Instance Segmentation Approach. Retrieved from https://debuggercafe.com/lane-detection-using-mask-rcnn/

[2] Ultralytics. (2024, February 21). YOLOv9: A Leap Forward in Object Detection Technology. Retrieved from https://docs.ultralytics.com/models/yolov9/

[3] Ke, B., Obukhov, A., Huang, S., Metzger, N., Caye Daudt, R., & Schindler, K. (2023). Marigold: Repurposing Diffusion-Based Image Generators for Monocular Depth Estimation [Computer software]. Retrieved from https://github.com/prs-eth/Marigold

[4] Zhou, X., Girdhar, R., Joulin, A., Krähenbühl, P., & Misra, I. (2022). Detecting Twenty-thousand Classes using Image-level Supervision [Computer software]. Retrieved from https://github.com/facebookresearch/Detic

[5] Ruhyadi, D. (2022). YOLO3D: YOLO 3D Object Detection for Autonomous Driving Vehicle [Computer software]. Retrieved from https://github.com/ruhyadi/YOLO3D

[6] Brazil, G., Kumar, A., Straub, J., Ravi, N., Johnson, J., & Gkioxari, G. (2022). Omni3D: A Large Benchmark and Model for 3D Object Detection in the Wild [Computer software]. Retrieved from https://github.com/facebookresearch/omni3d

[7] Jhawar, A. (2019). Traffic Sign Detection Training using YOLOv3 [Computer software]. Retrieved from https://github.com/aakashjhawar/traffic-sign-detection

[8] Lin, J., Zeng, A., Wang, H., Zhang, L., & Li, Y. (2023). One-Stage 3D Whole-Body Mesh Recovery with Component Aware Transformer [Computer software]. Retrieved from https://github.com/IDEA-Research/OSX

[9] Teed, Z., & Deng, J. (2020). RAFT: Recurrent All-Pairs Field Transforms for Optical Flow [Computer software]. Retrieved from https://github.com/princeton-vl/RAFT

Fig. 39: Identification of Parked Vehicles and direction of motion for moving vehicles using Optical Flow

Fig. 40: Identification direction of motion for moving vehicles using Optical Flow

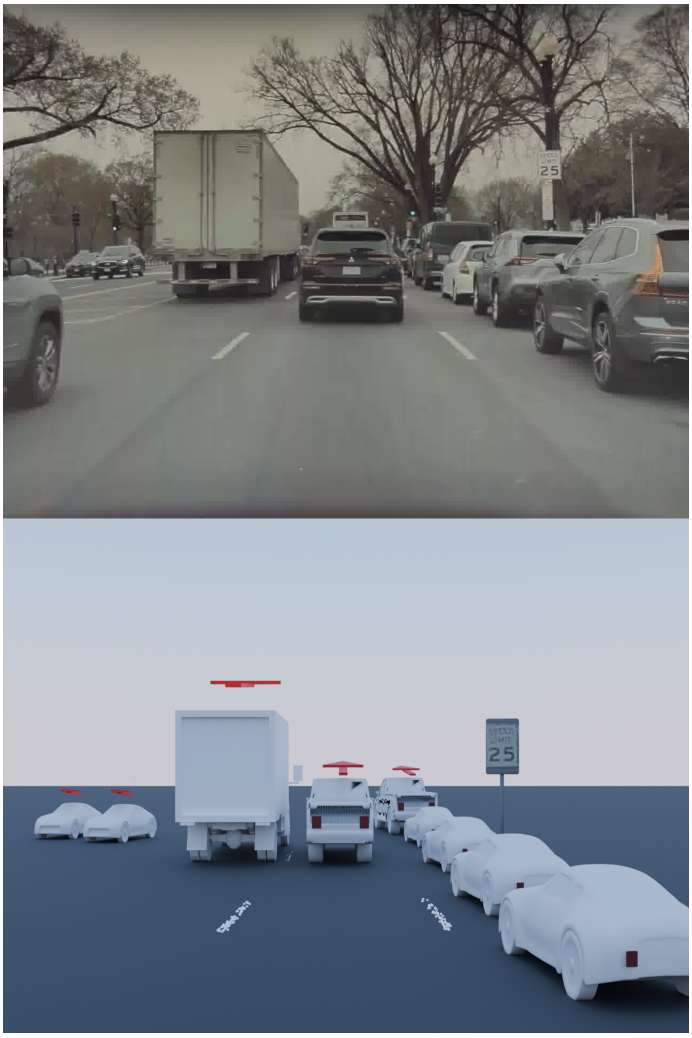

Fig. 41: Blender rendering of parked cars and moving cars using arrows for direction

Fig. 42: Blender rendering of parked cars and moving cars using arrows for direction

Fig. 43: Preprocessing the detected region of interest with denoising and grayscale conversion to sharpen and enhance the text visibility for OCR.

Fig. 44: Extracting text from detected signs to check for speed bump

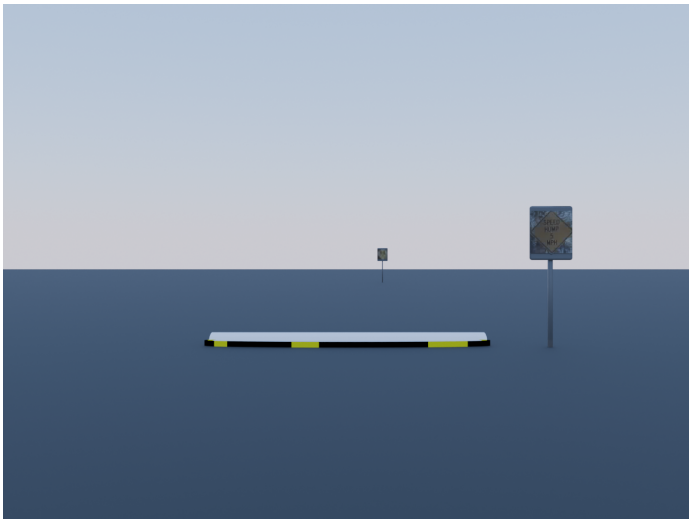

Fig. 45: Rendered output of a Speed Bump in Blender

APPENDIX-BLENDER VISUALIZATIONS

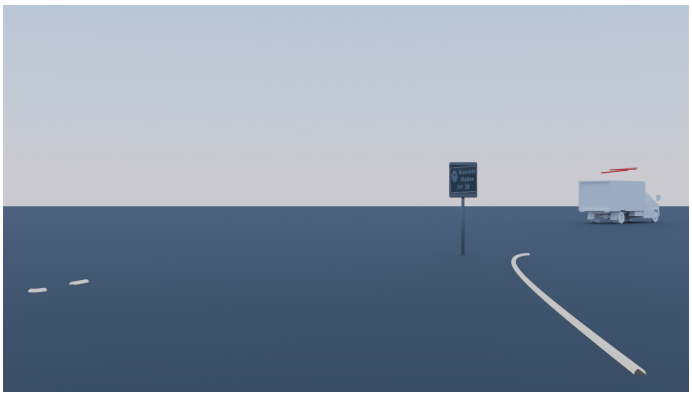

Fig. 1: Blender Visualization of Curved Lanes

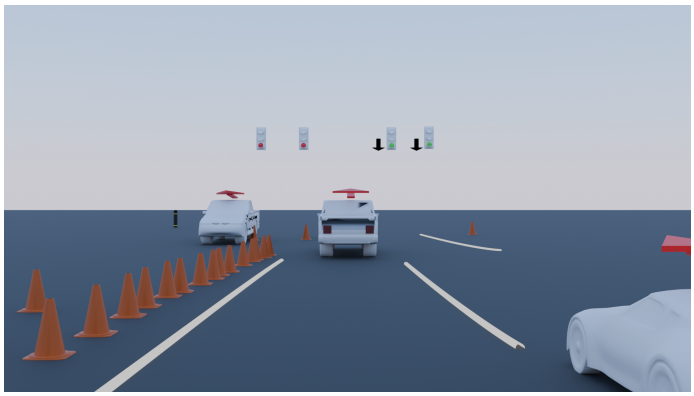

Fig. 3: Blender Visualization of Cones, Moving Vehicles, Traffic Light Color & Arrow Detection

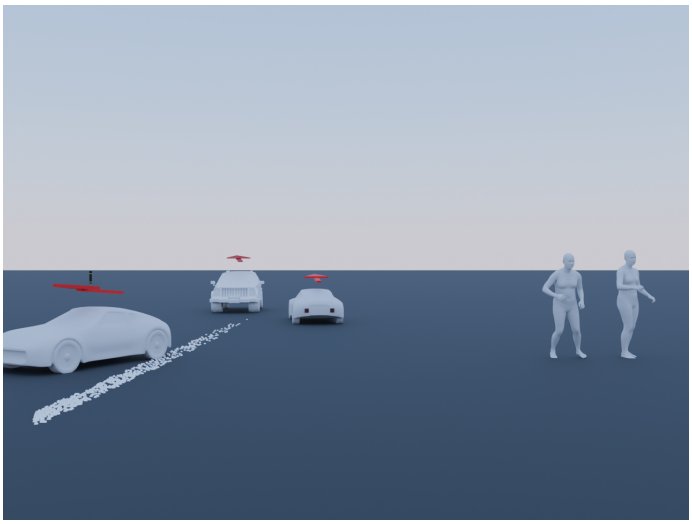

Fig. 2: Blender Visualization of Walking Pedestrians

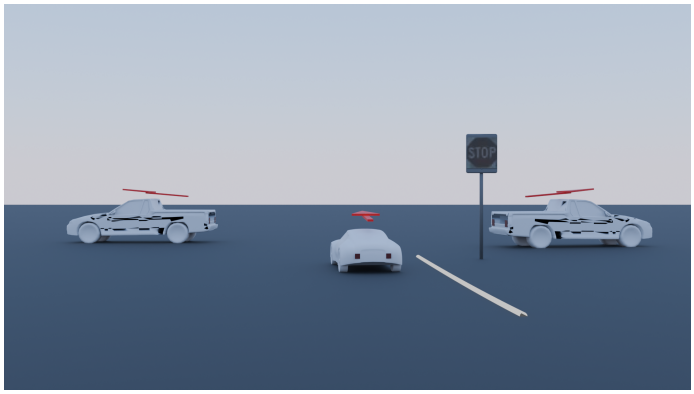

Fig. 4: Blender Visualization of Stop Sign

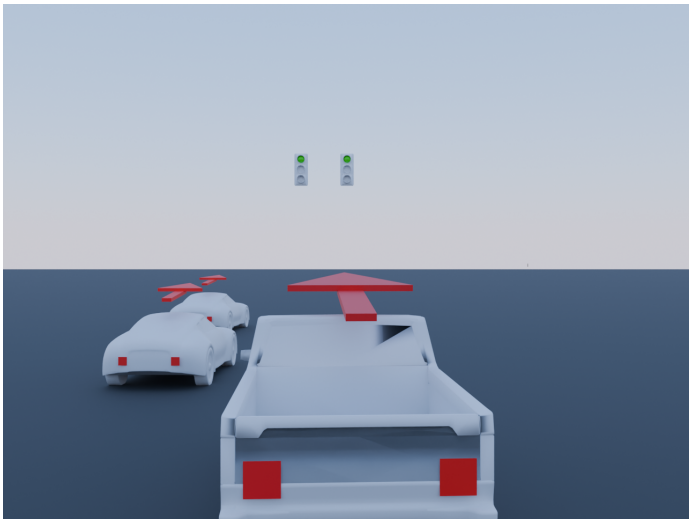

Fig. 5: Blender Visualization of Brake Lights

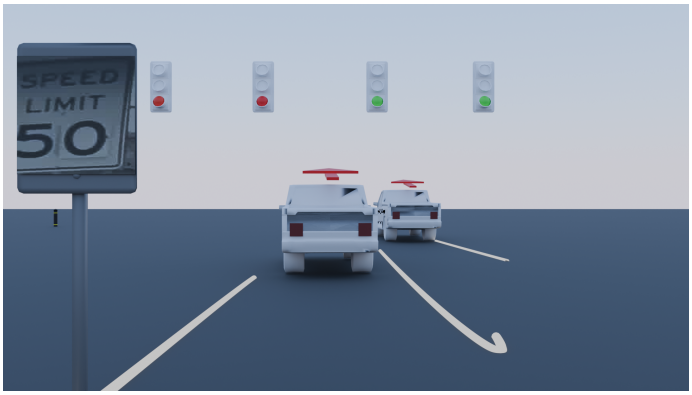

Fig. 6: Blender Visualization of Speed Limit Sign

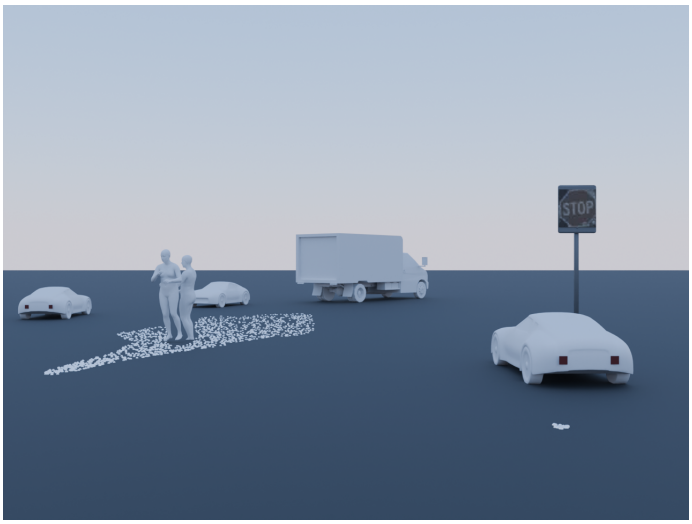

Fig. 7: Blender Visualization of Pedestrian

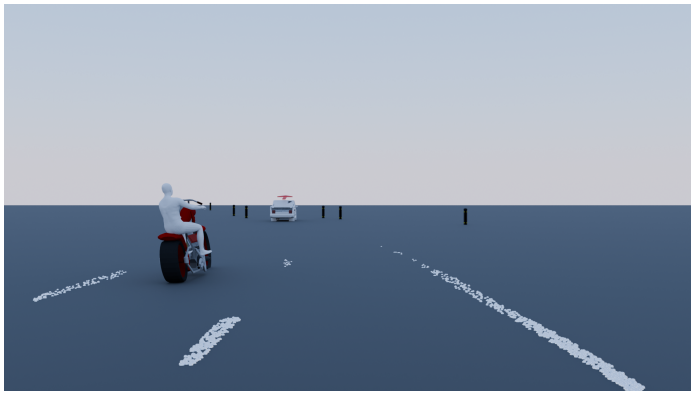

Fig. 8: Blender Visualization of Motorcycle

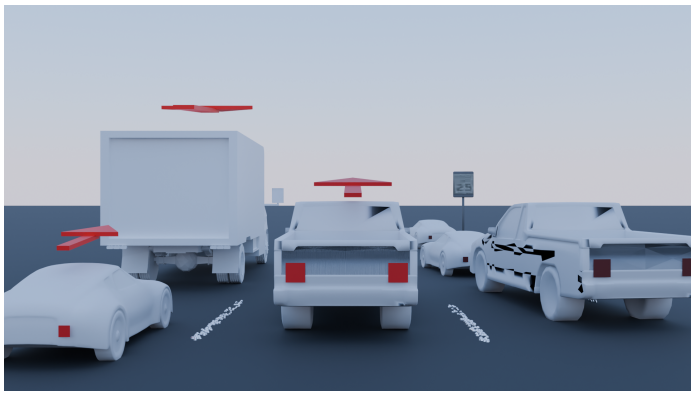

Fig. 9: Blender Visualization of Brake Lights, Parked & Moving Vehicles

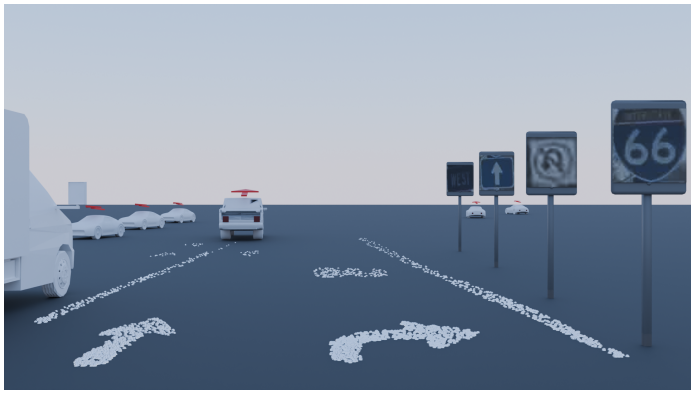

Fig. 10: Blender Visualization of Road & Street Signs

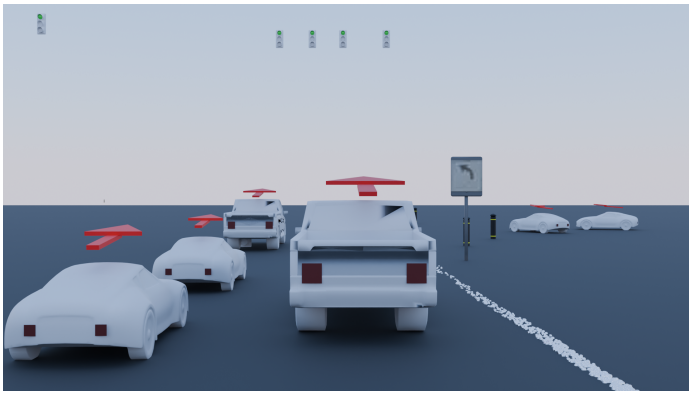

Fig. 11: Blender Visualization of Traffic Light

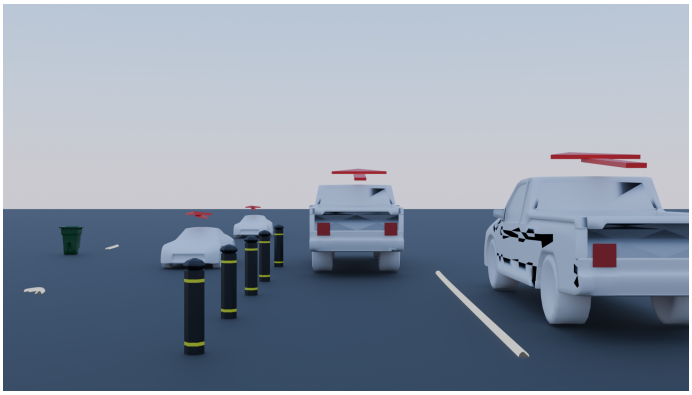

Fig. 12: Blender Visualization of Poles & Trash Can

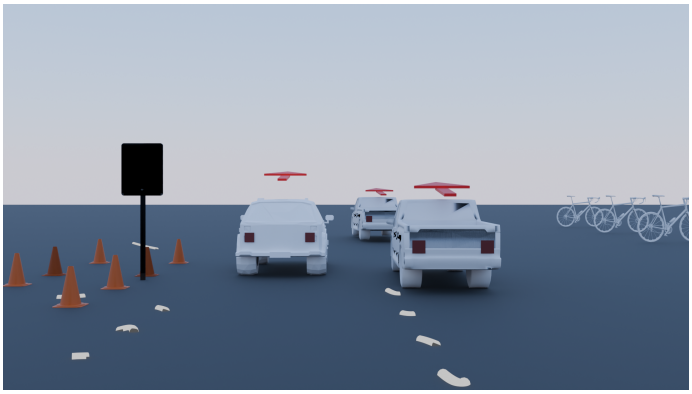

Fig. 13: Blender Visualization of Poles & Bicycle

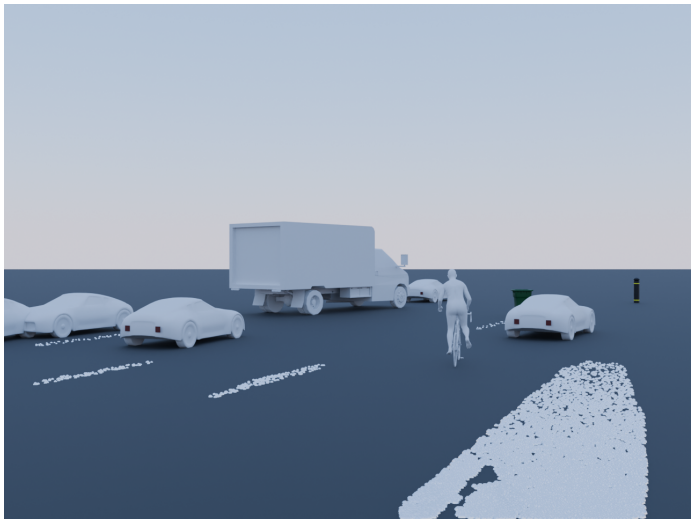

Fig. 14: Blender Visualization of Vehicles Stopped in Traffic

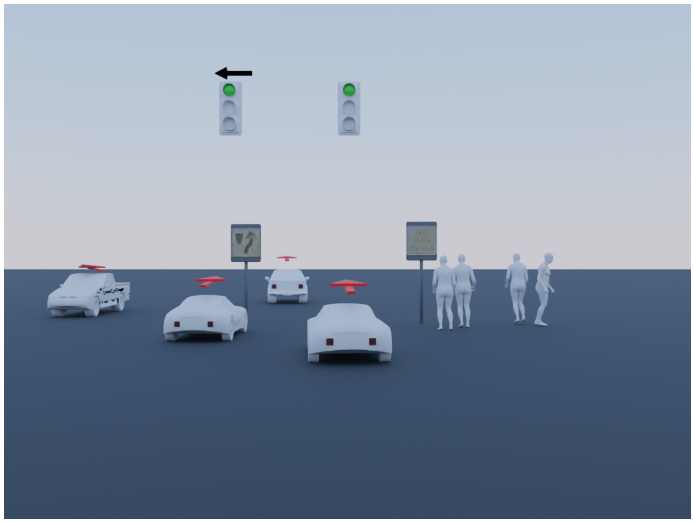

Fig. 15: Blender Visualization of Pedestrians and Traffic Light Arrow Classification