Einstein Vision

Computer Vision (RBE549) Project 3

Hrishikesh Pawar
MS Robotics Engineering
Worcester Polytechnic Institute
Email: hpawar@wpi.edu
Using 1 late day

Tejas Rane
MS Robotics Engineering
Worcester Polytechnic Institute
Email: turane@wpi.edu
Using 1 late day

Abstract—**In this project, we aim to develop a visualization system inspired by Tesla's dashboard display. Utilizing video data captured from the cameras of a 2023 Tesla Model S, our system processes and renders visualizations highlighting key elements essential such as lanes, vehicles, pedestrians, traffic lights, and road signs.**

I. Phase 1: Basic Features

The primary objective of Phase 1 is identification of essential features within a given scene. This phase concentrated on the recognition of the following elements:

- **Lanes:** Identifying different kinds of lanes on the road, which could be dashed, solid and/or of different color (white and yellow).
- **Vehicles:** Identifying all cars.
- **Pedestrians:** Identifying and locating pedestrians and displaying them in the scene.
- **Traffic Lights:** Indicate the traffic signals and it's color.
- **Road Signs:** Identifying the road signs, primarily the stop sign.

The initial step of our processing pipeline involves subsampling of undistorted video stream, wherein one key frame is extracted for every 5 frames. These key frames, derived from various video sequences, are then utilized as inputs to the models.

Following steps were taken to execute the phase.

A. Depth Estimation from Monocular Images

The first step in the entire processing pipeline is to perform depth estimation from monocular RGB images. We use Marigold[1] for this task. Marigold generates relative depth from monocular RGB images, which we scale to a factor such that objects spawned in Blender look visually correct.

B. Lane Detection:

Lane detection was primarily conducted using the lanenet model, which excels at identifying lane markings in images. Despite its efficacy in detecting lanes, CLRNet lacks the capability to classify the type of each lane. Our initial approach was to generate masks from CLRNet and classify them using DBSCAN. However, this approach proved to be not generalized. In our search for a better solution, we transitioned to a

[1]Marigold

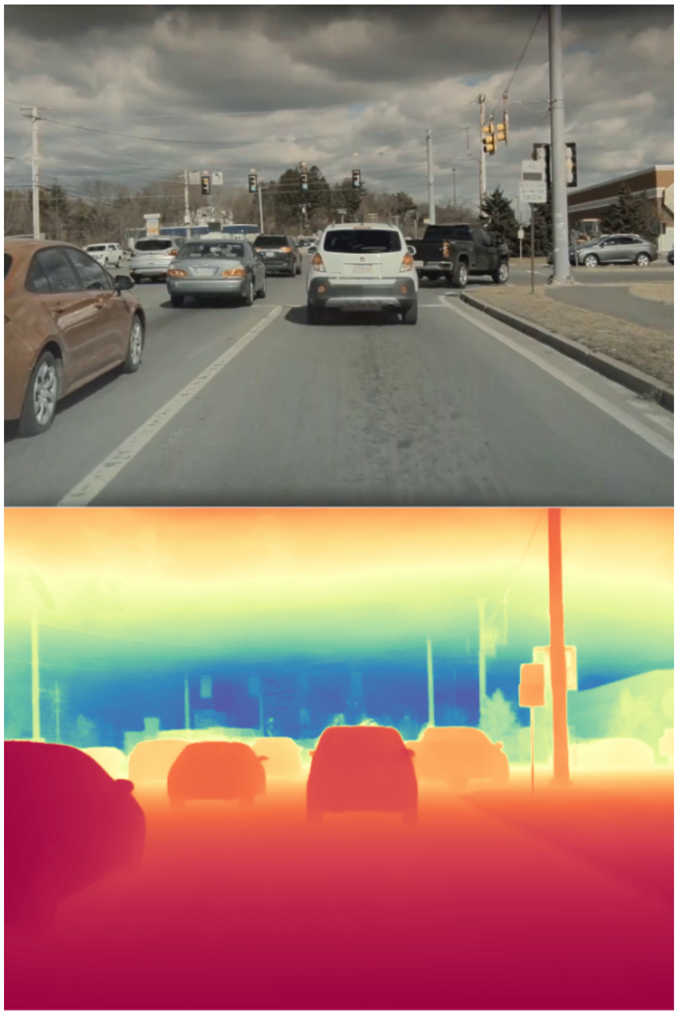

Fig. 1: Depth Estimation from Monocular RGB Image. Top shows the original image and Botoom shows the estimated depth.

Mask RCNN model, which detected and classified lanes into six different classes as follows:

- Divider line
- Dotted line
- Double line
- Random line

- Road sign line
- Solid line

One observation we made was that the model provided accurate results for most cases. However, it also produced some false positives, as illustrated in Fig. 2 and Fig. 3.

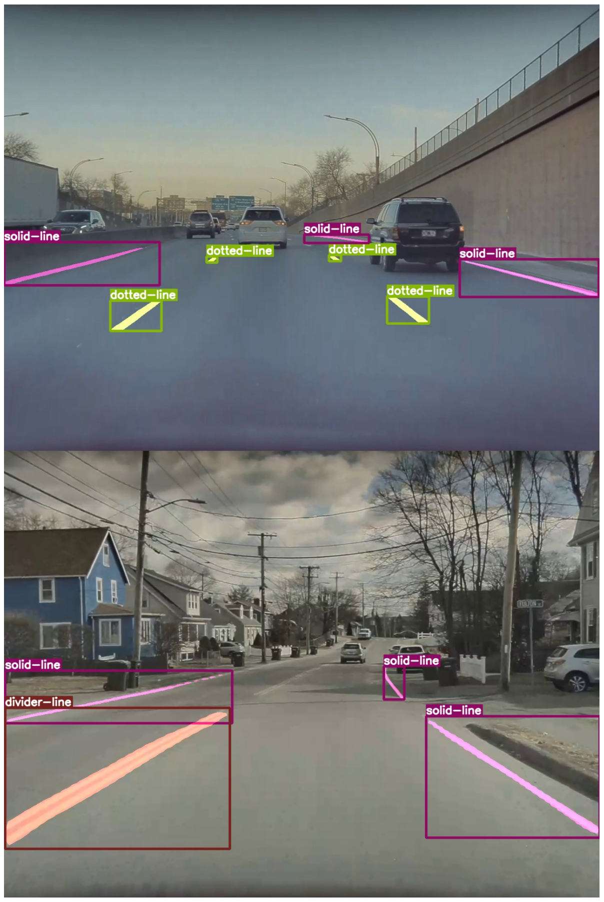

Fig. 2: Lane Classifications Overview

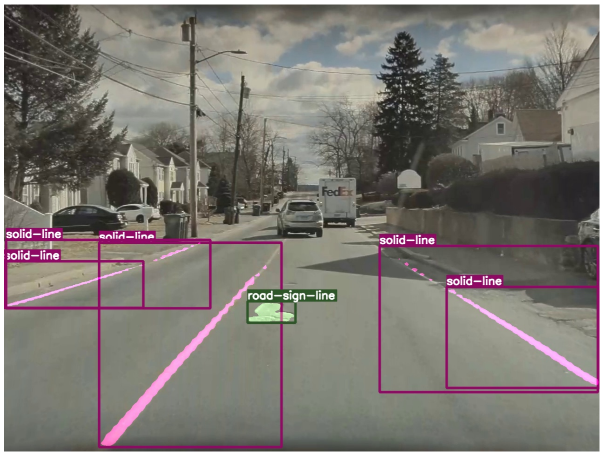

Fig. 3: False Positive Detection

From the detected masks, our approach was to extract control points for lane depiction. Specifically, we sampled on the nth row across the image to determine the mean position of lane markings within that row. This is achieved by iterating over the binary mask of a detected lane at a specified sampling rate. For each sampled row, we calculate the mean column index of pixels classified as part of the lane. These mean positions, representing averaged horizontal locations of lane markings, serve as control points as seen in Fig 4. These control points were then utilized as input for plotting Bezier curves in Blender, providing a representation of lane paths.

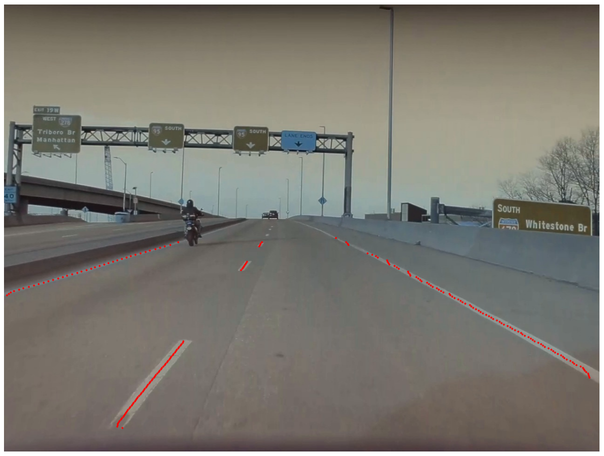
Fig. 4: Lane Control Points

C. Vehicles, Pedestrians, and Traffic Lights Detection

The detection and segmentation of vehicles, pedestrians, and traffic lights were accomplished using the YOLOv8 model. The model was used to to recognize five vehicle subcategories: *Car*, *Bus*, *Truck*, *Bicycle*, and *Motorcycle*. Similarly, the same model was used to detect pedestrians, traffic lights, and road signs. The outcomes of these detections can be seen in Fig. 5 and Fig. 6.

D. Integration and Visualization in Blender

The detections from both the Mask RCNN (for lane detection) and YOLOv8 models were aggregated into a unified JSON file format. This file encapsulates the coordinates of detected objects, which are subsequently transformed into Blender's coordinate framework. Specifically, for the Yolov8 model the centroids of the bounding boxes, indicative of the detected objects' locations, were converted and used as spawning points within Blender.

II. Phase 2: Advanced Features

Building upon Phase 1, Phase 2 aimed to introduce a higher level of granularity. This phase focuses on the detailed classification and subclassification of detected entities, the incorporation of orientation data, and the visualization of these elements within the rendered environment.

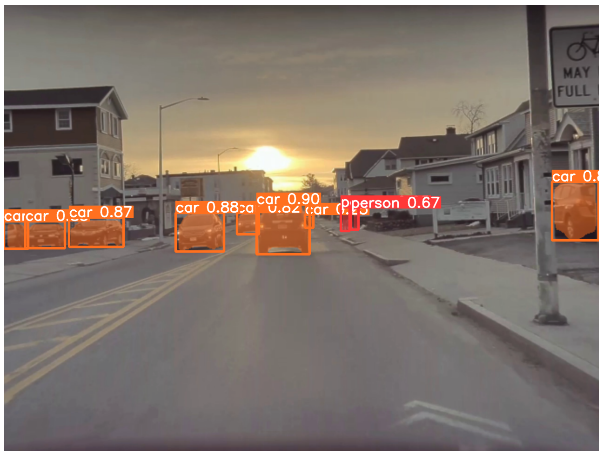

Fig. 5: YOLOv8 Object detection Outputs

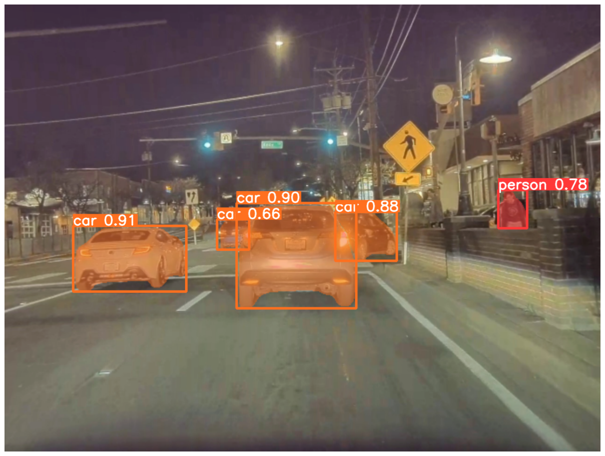

Fig. 7: YOLOv8 Object detection Outputs

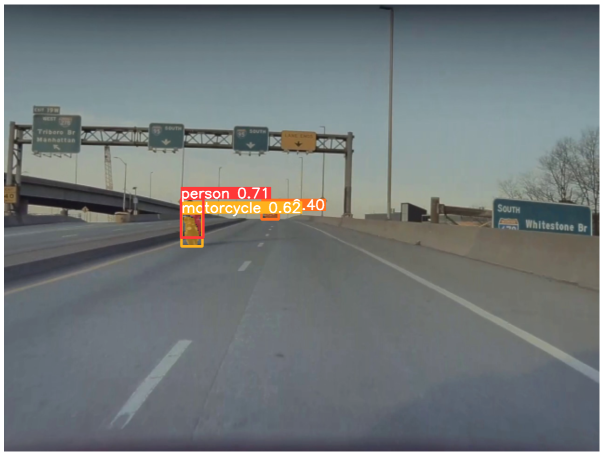

Fig. 6: YOLOv8 Object detection Outputs

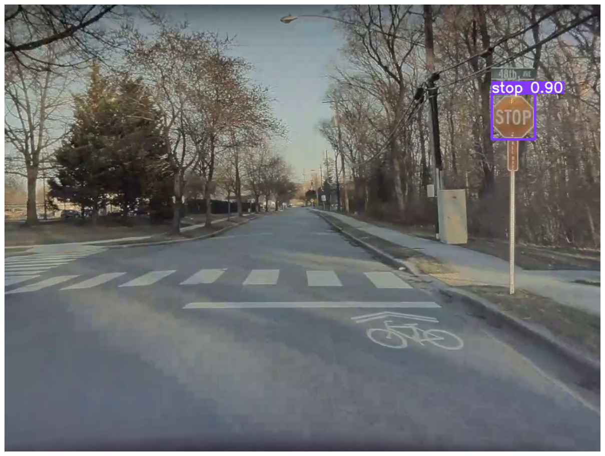

Fig. 8: YOLOv8 Object detection Outputs

A. *Vehicles*

A classification regime was introduced to categorize vehicles not only by type but also by specific subtypes. Additionally, the orientation of each vehicle is determined. This is done by estimating the 3D bounding box of the vehicle

B. *Traffic Lights*

Building upon the initial identification of traffic lights, this phase incorporates the classification of directional arrows present within traffic lights. We use YOLOv8 for this task.

C. *Road Signs*

In addition to the recognition of stop signs identified in the previous phase, this phase extends to include the detection of ground-level road signs, such as directional arrows and speed limit indicators. We use YOLOv8 trained on the GLARE dataset[2] for this task.

[2]Glare Dataset

D. *Additional Objects*

The scope of detection is further expanded to encompass various urban infrastructure elements, including dustbins, traffic poles, cones, and cylinders. These objects are detected and segmented using Detic[3].

E. *Pedestrian Pose*

Identification of pedestrian pose was performed using OSX[4]. OSX takes the RGB image as input and directly generates the Blender meshes as output. The generated mesh files are then inserted in the corresponding scenes and rendered. Fig 11 shows the original RGB image and the generated Blender mesh files.

III. Phase 3: Bells and Whistles

This phase dives deeper into adding more cognitive abilities for better decision making in the path planning stage of our self-driving car. Mainly, we focus on understanding various attributes of the vehicles in the environment, like the status of

[3]Detic
[4]OSX

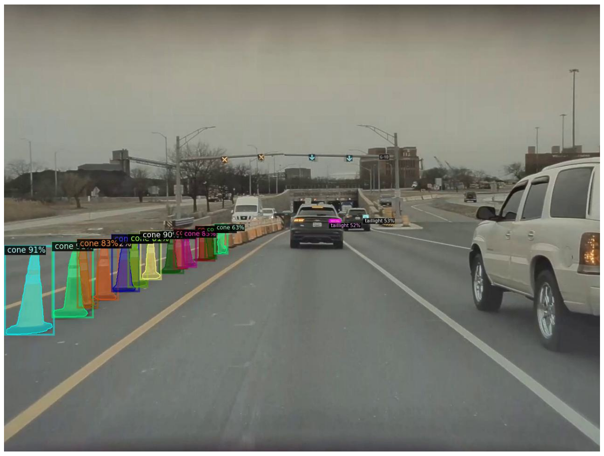

Fig. 9: Detic Object detection Outputs (cones)

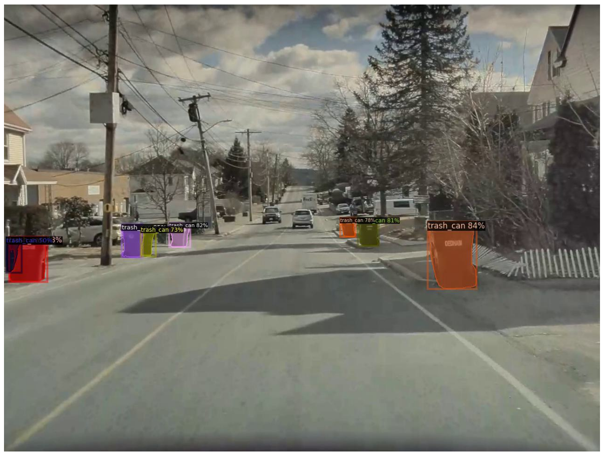

Fig. 10: Detic Object detection Outputs (trash cans)

their brake lights and turn indicators, as well as whether they are moving or parked.

A. Brake lights and indicators of the other vehicles

The brake lights of the vehicles are detected and segmented using Detic[5]. Then, we perform post-processing on these segmented sections of the image to identify whether the brake lights and turn indicators are *on* or *off*. The post-processing steps are mentioned as follows:

- The first step is to classify the detected brake lights as *on* or *off*. This is done by converting the segmented patches to `YCrCb` color space and applying adaptive threshold on the `Y` (luminance) channel.
- Then, the classified brake lights are assigned to the corresponding vehicles, based on whether the detected brake light lies within the detection bounding box of the vehicle.
- Now, there are a few cases that might arise here. Based on the accuracy of the detection model, there can be either none, one or both brake light detection for the vehicle.

[5]Detic

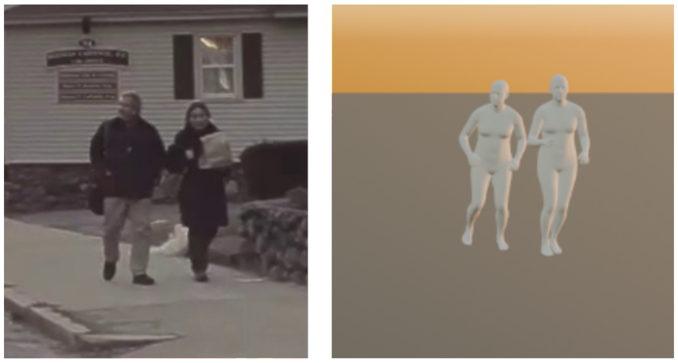

Fig. 11: Original RGB image showing humans, and their corresponding generated Blender meshes.

- In case of no detection, we assume that the brake lights and indicators of the vehicle are off.
- In case of only one detection for the vehicle, depending on the status of the detected brake light (*on* or *off*), we assume both the brake lights to be *on* or *off*.
- In case of both the brake lights being detected for the vehicle, we can figure out all the possible cases. If both are *on*, the brake lights are *on*, and vice versa. In case only one of them is *on*, the vehicle has either the *turn left* or the *turn right* indicator active.

B. Parked and moving vehicles

To differentiate between parked and moving vehicles, we utilized optical flow analysis using the RAFT[6] algorithm, which generates monocular optical flow images showing relative flow between pixels. Increased flow rates are depicted by intensified colors in these images. Initially, we tried using the Sampson distance to identify moving vehicles based on their higher flow rates. However, we faced a challenge where vehicles in front, moving relatively slower compared to our vehicle, were incorrectly identified as parked due to their lower flow rates. To tackle this issue, we developed a multi-step approach. Firstly, we assessed the net flow within each bounding box provided by Detic[7] and compared it with the flow from neighboring regions. If the flow difference was minimal, indicating limited movement of the object relative to the scene, and if the net flow of the mask itself fell within a specified range (suggesting that cars moving in the same direction were below a certain threshold), we classified the vehicle as parked. Additionally, to account for our vehicle's motion, we employed flow subtraction using a Gaussian mask, with the mask's variance proportional to the variance of flow in the image. This method enabled us to more accurately distinguish between parked and moving vehicles.

IV. Conclusions and Discussions

In this project, we get a taste of implementing the dashboard visualizations for a self driving vehicle. We use various pre-trained models for self-driving tasks like depth estimation,

[6]RAFT
[7]Detic

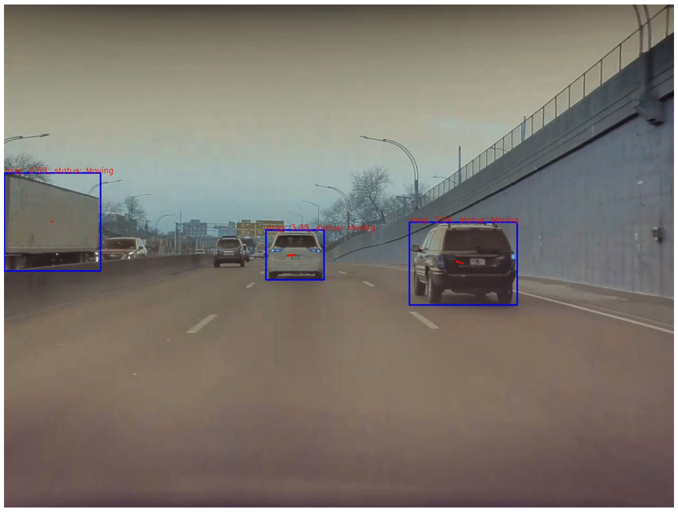
mag: 0.59, status: Moving
mag: 3.38, status: Moving
mag: 3.54, status: Moving

Fig. 12: Detected parked and moving cars, with their motion directions

Vehicle and road-side objects detection, lane classification, human pose estimation and optical flow on the data provided from a Tesla Model S. Based on the detection and segmentation results of all these models, the corresponding objects are spawned in Blender so as to simulate the movement of these objects in the environment as the self driving car moves through it. For this project due to interest of time we had to rely on existing pre-trained models. However, if this needs to be deployed on an actual self-driving car in the future we must train our own custom model with a large dataset, which can perform multiple such tasks in parallel. The Hydra-Net architecture deployed by Tesla is one such example, where the model shares a common backbone (feature extractor) and there are multiple heads to segment and detect multiple objects in parallel.

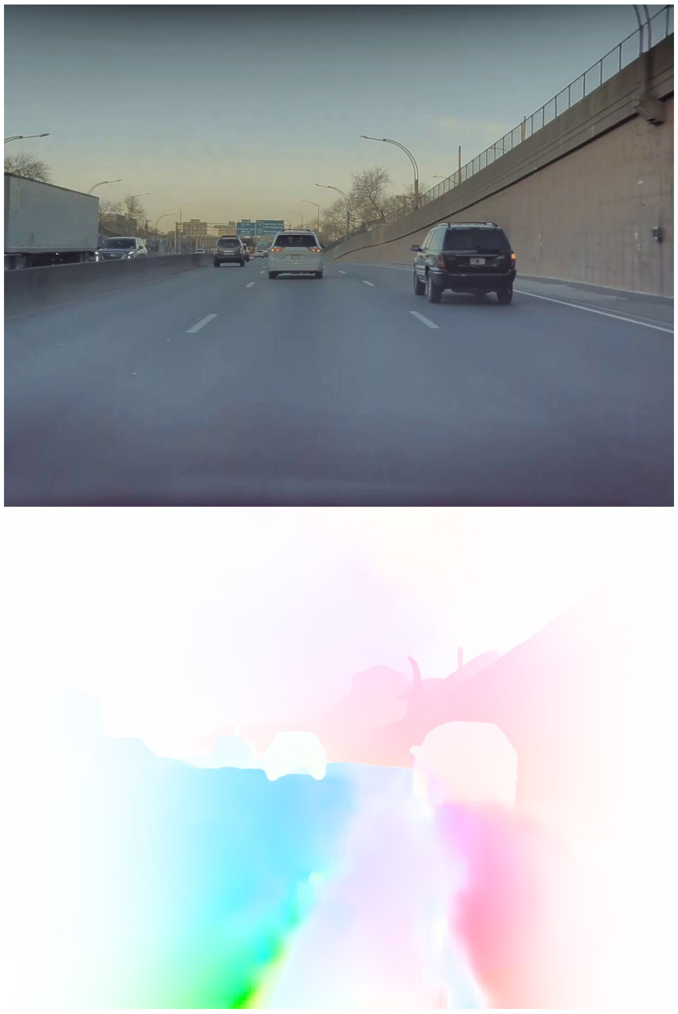

Fig. 13: Original Image, and its optical flow.

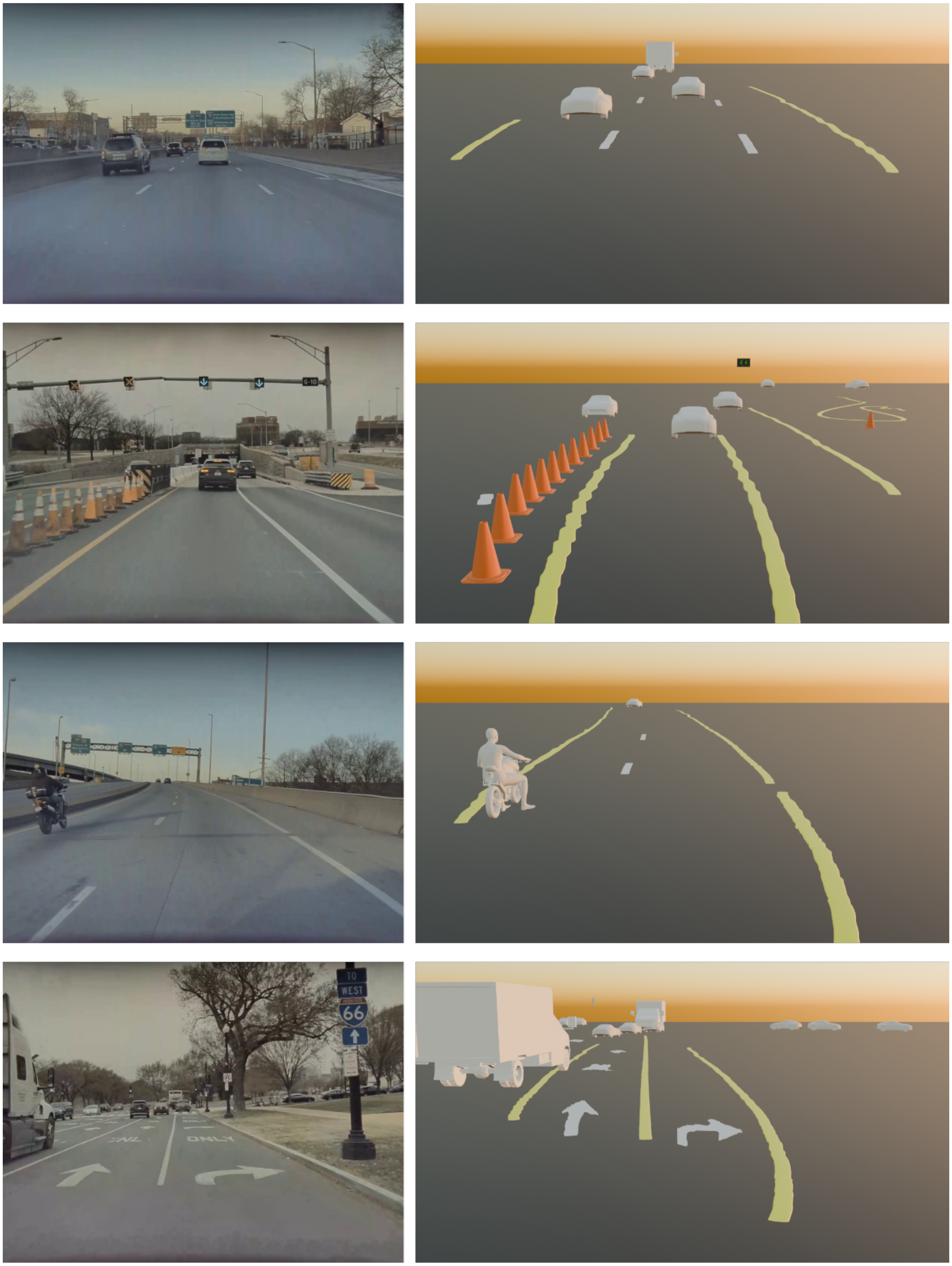

Fig. 14: Rendered Output of different scenarios (1/3). From Top: (1) Generic Rendered scene, (2) Cones and Arrows on traffic lights, (3) Motor Cycle, (4) Signs painted on the road.

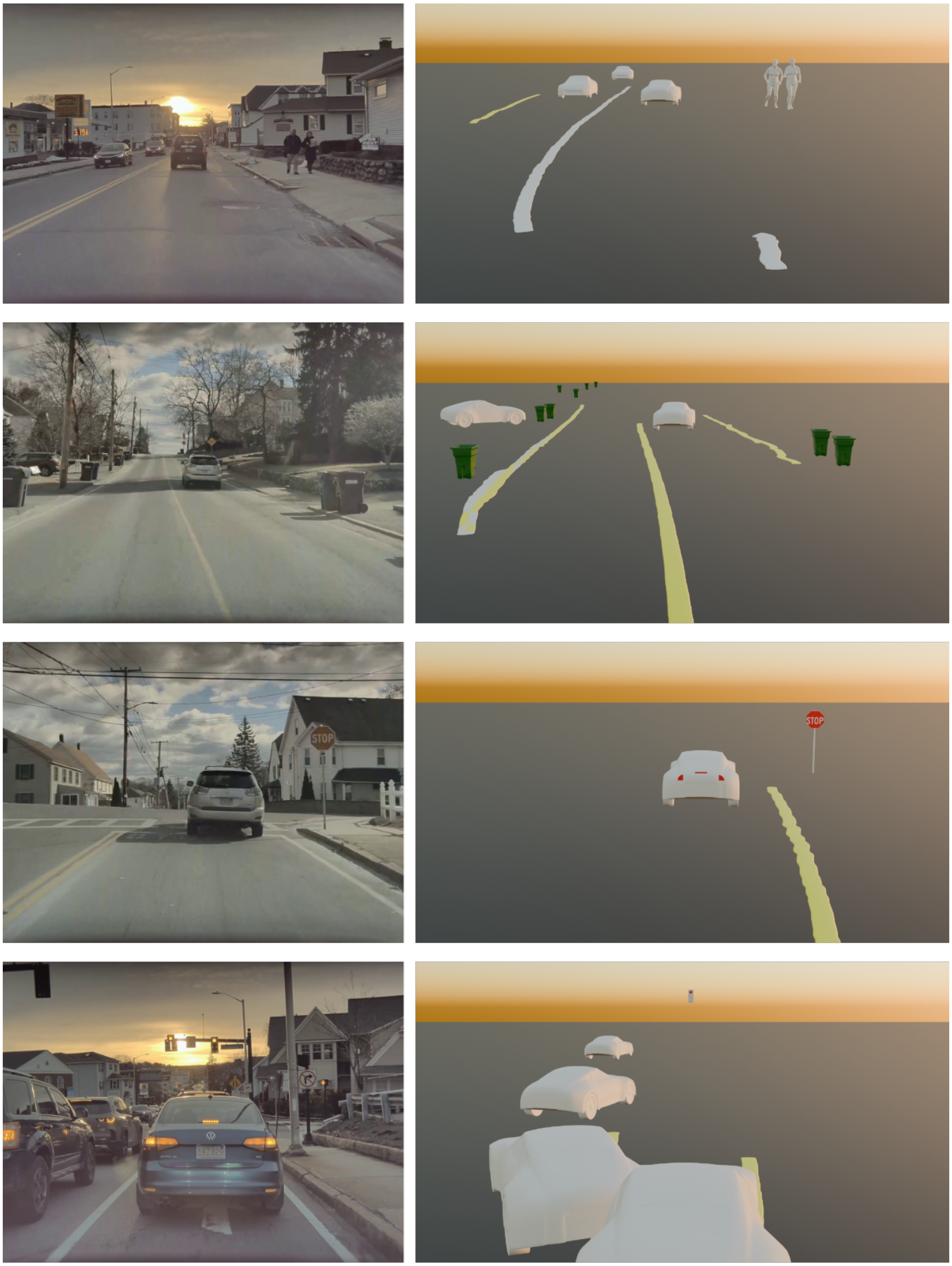

Fig. 15: Rendered Output of different scenarios (2/3). From Top: (1) Humans in different poses, (2) Trash cans, (3) Stop sign, (4) Traffic Light.

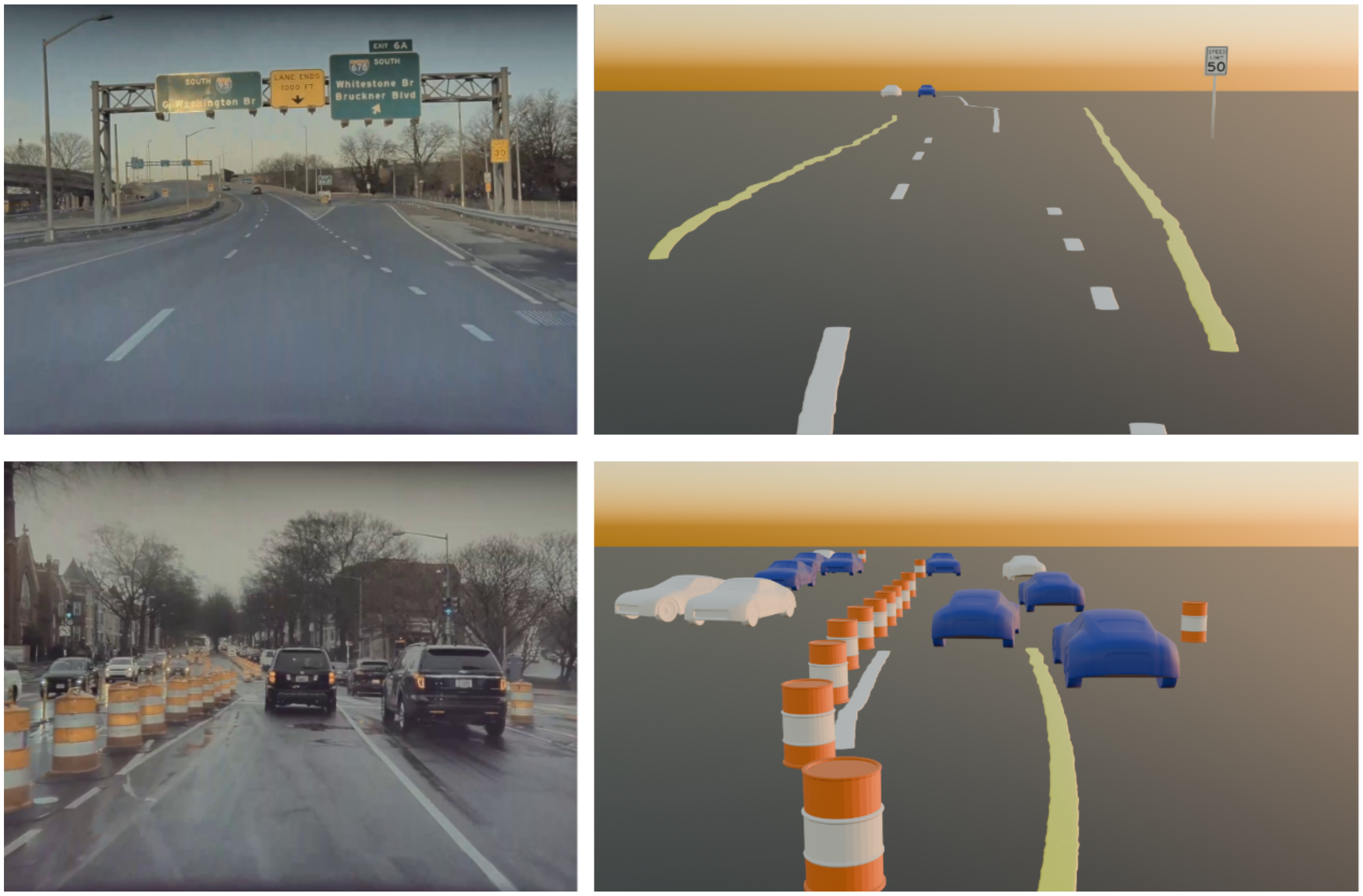
Fig. 16: Rendered Output of different scenarios (3/3). From Top: (1) Different types of lanes, (2) Traffic Cylinders.