RBE 549 Computer Vision Project 3 - Einstein Vision

Kaushik Kavuri Subrahmanya
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: ksubrahmanya@wpi.edu

Butchi Adari Venkatesh
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: badari@wpi.edu

***Abstract*—An important aspect of any computer vision project is the visualization of the output. In this project, we aim to create a visualization of the onboard camera views of a Tesla car. We achieved this by extracting various objects from the video and rendering them through Blender.**

I. Data preparation

Undistorted videos from the onboard cameras were provided, which we are using as the basis for this project.The work on this project is divided into three parts, with each part adding more features. To process the videos, we first extracted frames from the videos by using OpenCV2. As the number of frames are high, and processing them would not be a feasible task, we reduced the count of frames by 10 times by selecting only every 10th frame. We also used OpenCV2 to calculate the front camera's calibration matrix.
After extracting this, for each vehicle, we created an alternative asset with visible red braking lights. Using the given Traffic Light asset, we created 3 different traffic lights for red, yellow, and green traffic lights. We also created a simple elongated cylinder to resemble a speed hump.

II. Object Detection

In this phase, we implemented the detection of lanes, vehicles, pedestrians, traffic lights, and road signs. We first calculate the depth of each part in a frame using Zoe Depth. Zoe Depth is a deep learning model that gives the output of depth data in a grey-scale image, where a completely white pixel corresponds to a close object, and vice versa.

1. **Lane Detection**: Lanes were detected using CLRerNet. Using this model, we get the points that correspond to a lane. Further, we used a segmentation model (Masked RCNN) that classifies the lanes into solid, dashed, and double lines. After this, we took the points and found the points which lie inside the bounding box of the classified segmentation bboxes. The depth of each point is then calculated using Zoe Depth to render in Blender.

2. **Vehicle Identification**: To detect the vehicle object on the road, we use YOLO-World. YOLO-World is a real-time Open-Vocabulary object detection model that supports prompt tuning to detect classes. Using YOLO-World, we get a bounding box for each type of vehicle. To detect a vehicle's orientation, we use YOLO-3D.

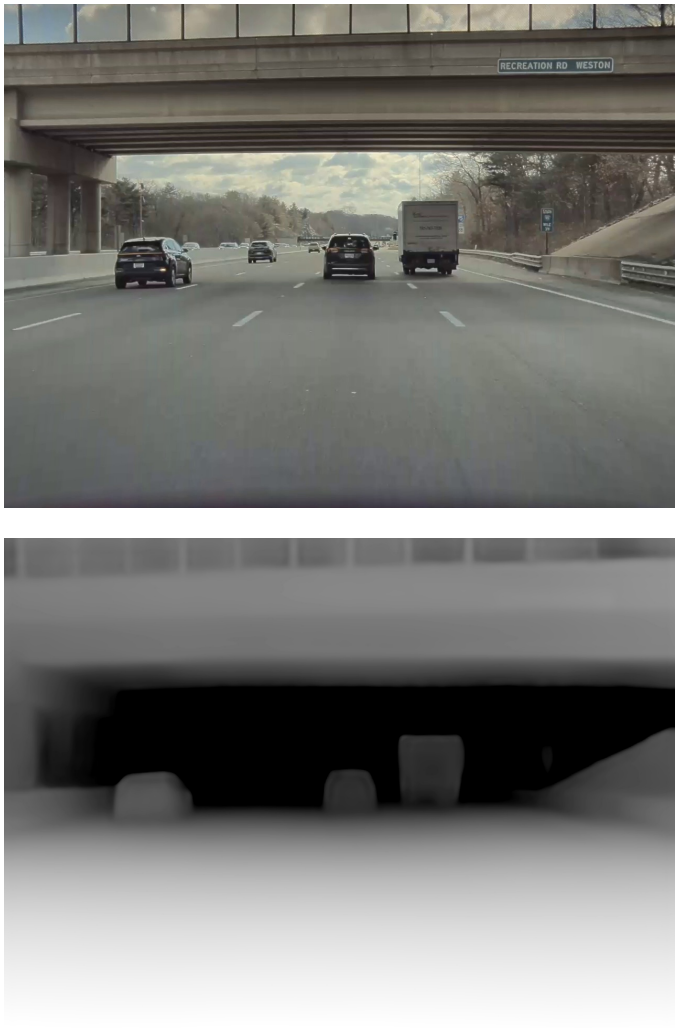

Fig. 1: Scene 1 - Input, ZoeDepth Output

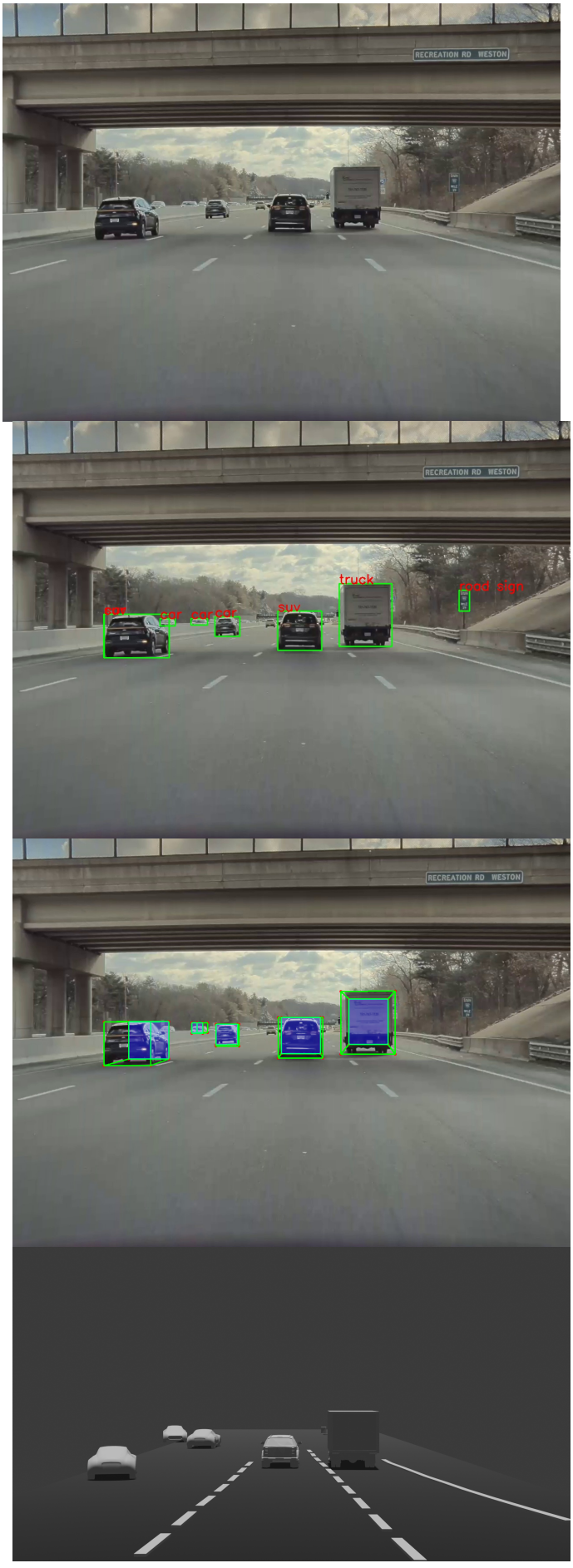

Fig. 2: Scene 1 - Input, YoloWorld, Yolo3D outputs, Final

3. **Pedestrians**: To detect pedestrians, we used PyMAF which directly gives us a blender mesh object(.obj file). The location is obtained as a bounding box and the corresponding depth of the person is then taken from Zoe depth. This model consumes a lot of time and VRAM to compute, but the smpl human mesh has the orientation and pose of the pedestrian included in it. This mesh and the depth are then rendered through Blender.

Fig. 3: Scene 8 Pedestrian Input, Output

4. **Traffic Lights**: We detected traffic lights using YoloWorld and Yolo2d. The procedure, issue, and fix used were the same as for vehicle type identification. The color of the traffic lights was also detected using YoloWorld. The bbox was used to calculate the location of the light which enables us to get depth info from Zoe Depth. The corresponding colored traffic light assets were then rendered through Blender.

5. **Road Signs**: Road signs such as Stop signs and speed limit signs were detected through YoloWorld. Depth from Zoe Depth was used to render the signs. We used EasyOCR to detect the speed limit from the cropped bbox image of the speed sign.

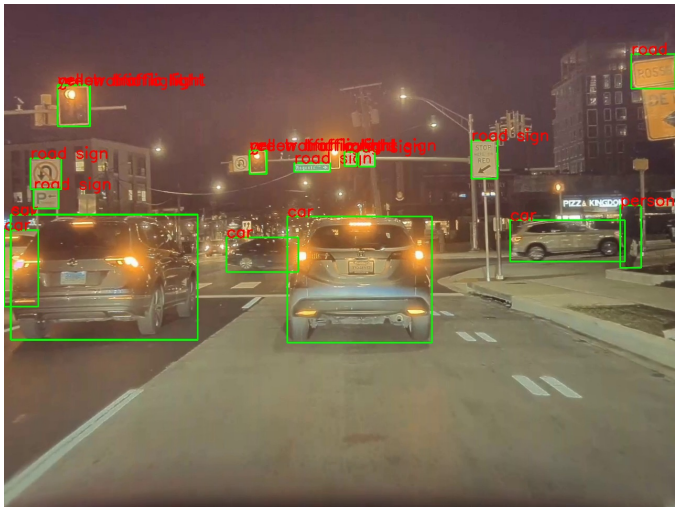

Fig. 4: Scene 10 Traffic Sign, Road Sign YOLO World Output

6. **Road Objects**: Road objects such as traffic cylinders and dustbins were detected through YoloWorld. Depth from Zoe Depth was used to render the objects.

Fig. 5: Scene 5 - Pedestrian, Trash can, Road Sign YoloWorld output, Final

7. **Brake Light Indication**: To detect the brake lights, we used tail-light-detection model which was trained based on YOLOv8. The model detects the status of the brake light and returns a label on whether the brake light was engaged or not. If the light was engaged, the model used for a vehicle was replaced with the corresponding alternative asset we created.

8. **Stationary/Moving Vehicles**: We used RAFT for optical flow and Sampson distance from OpenCV2 to distinguish between stationary and moving vehicles. Stationary cars are rendered as green cars in our Blender renders.

III. Extra Credit

A. Speed Bumps

To detect speed bumps, we used YoloWorld to detect a road sign. We then used EasyOCR on the speed sign. If the output included the term 'hump', we placed a speed hump asset near the road sign, on the road.

IV. Challenges and Workarounds

A. Scaling in Blender

As all assets in individual blender asset files were of vastly different sizes, placing them all and rendering them in a single scene resulted in completely weird scaling. We solved this by applying a custom scaling on each blender object to bring all objects to the same relative scale. Orientation was also fixed this way by applying Euler rotations on them.

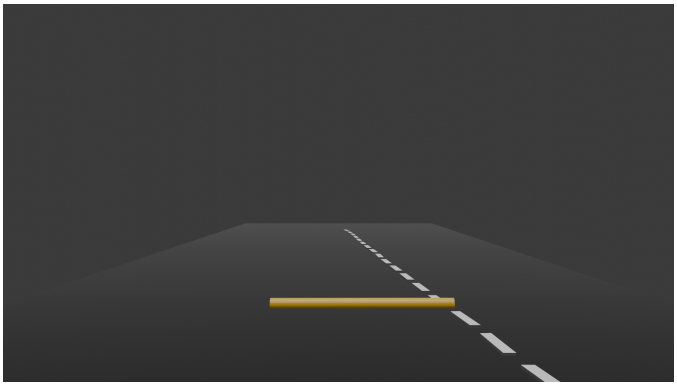

Fig. 6: Scene 9 Speed Hump Blender Output

B. Camera Location

We had to change the camera location manually for a few scenes where the vehicles were sometimes rendered below the visible rendered images.

C. Vehicle Detection

One issue with Yolo3d was that the camera calibration matrix used by the model resulted in incorrect outputs where the bounding boxes were pointing in the wrong orientations. To solve this, we replaced the camera calibration matrix that we obtained using OpenCV2. After making this change, the 3D bounding boxes were pointing in the correct orientations. Another issue with this method was that the centers of bounding boxes of yolo3d and yoloworld were not equal. To solve this, we implemented a simple algorithm (with some similarities to non-max suppression) that calculates the Euclidean distance between the center from yolo3d to all the centers from yoloworld. The minimal distance distance is then taken as the same object/car. Using this, we have the vehicle type and their orientation.

D. Pedestrians

We first added an armature to the mesh provided and intended to use this rigged setup to render humans. We obtained the pose and location of the person through Yolov8 Keypoint detection. However, rendering the person proved challenging as we could not orient the person anatomically correctly. So, we abandoned this approach and used the above-mentioned approach.

E. Road Signs

One minor issue with this was with the orientation of the signs as they were pointing away from the view initially. This was resolved by applying an Euler rotation of π radians about the Z axis in the Blender script.

F. Stationary/Moving vehicles

Our 10-frame skip approach to data generation resulted in stationary cars being detected as moving in some situations, as the time skip between frames was a bit high. Another issue was that if our car speed and the speed of the car ahead were the same, the optical flow and Sampson distance both resulted in the car being detected as stationary. We applied a threshold to the Sampson distance but it wasn't always successful and the threshold needs to be tuned further than we could in the limited time.

V. Results

A few frames from our renders are shown below.

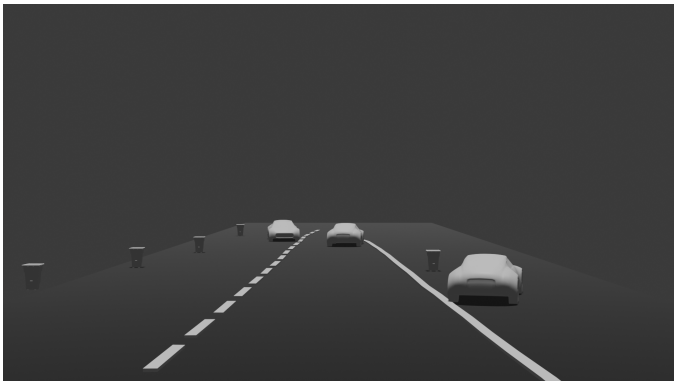

Fig. 7: Scene 5 Blender Output

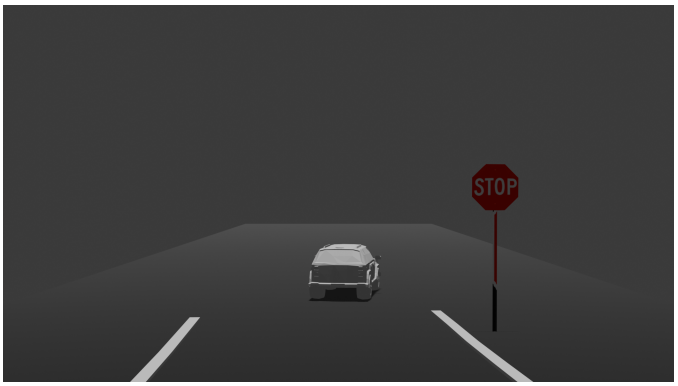

Fig. 8: Scene 5 Stop Sign Blender Output

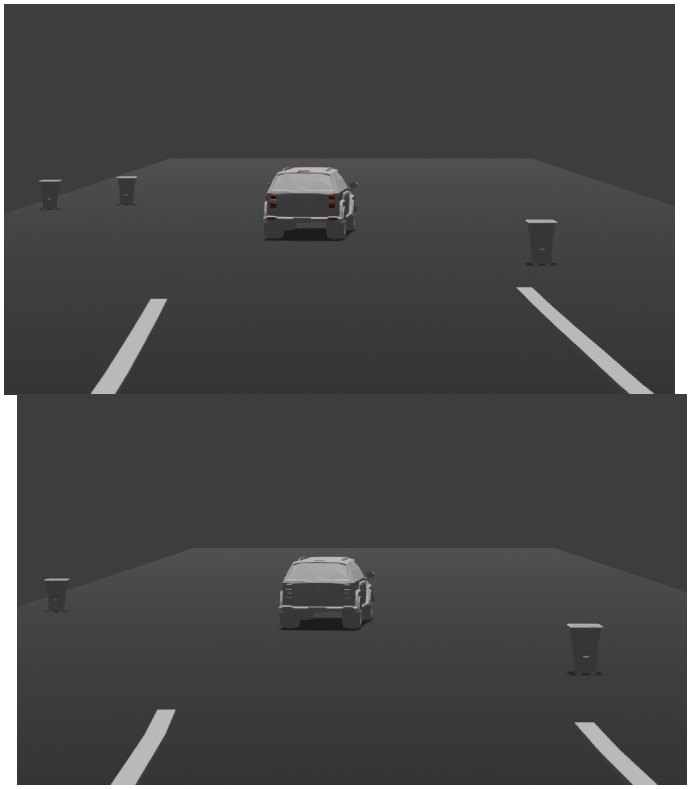

Fig. 9: Brake Light Demonstration

VI. References

1) `YOLO3D`: To Detect Bounding boxes and orientations of the vehicles.
2) `Yoloworld from ultralytics`: To Detect Bounding boxes of the types of vehicles, types of traffic lights, person, traffic cone, speed limit sign, road signs, speed breaker/hump, trash can. Traffic lights, Persons, Stop Sign.
3) `CLRERNet`: To detect Lane Detection. Detects (Solid Line, dashed Line, Double Line).
4) `PyMAF`: Generates human Mesh .obj files for every frames.
5) `EasyOCR`: To detect text on the road signs.
6) `ZoeDepth`: To detect depth of the frames.
7) `RAFT`: To get optical flow b/w each frames.
8) https://github.com/gsethan17/one-stage-brake-light-status-detection: To detect car rear lights.

Acknowledgment

The authors would like to thank the professor for the instructions and knowledge imparted via the course.