RBE549 Project 3 - Einstein Vision

Yaşar İdikut
yidikut@wpi.edu
Using 2 late days

Harshal Bhat
hbhat@wpi.edu
Using 2 late days

I. Introduction

In this project, we are tasked with creating 3D visualization for a car equipped with cameras. Of the four given cameras, we only use the front-facing one. We are given 1 minute-long 13 scenes with around 2000 frames for each of the scenes. Our approach for processing and rendering is per frame. We don't employ any object permanence or tracking logic. We rely on multiple object detection models and classical approaches for finding and reasoning about the images, and a metric depth estimation model to project the objects detected in the image in the coordinate frame of our rendering program (Blender).

In the next few sections, we will go over our methodology and results for the detection and vizualization of following tasks:

- Phase 1: Basic Features
  - Lanes
  - Vehicles (with pose and types, which was added at phase 2)
  - Pedestrians (with pose, which was added at phase 2)
  - Traffic Lights
  - Road Sign (Stop Sign)
- Phase 2: Advanced Features
  - Traffic Objects (Dustbins, Traffic Poles, Traffic Cones)
  - More Road Signs (Ground Arrows and Speed Limit Signs)
- Phase 3: Bells and Whistles
  - Break Lights and Indicators of the other vehicles
  - Stationary and Moving Vehicles

II. I. Phase 1: Basic Features

A. Lane Detection and rendering

We preprocess the image by resizing, padding, and normalizing the images to a uniform size and format suitable for analysis. YOLO Panoptic v2[1] is used for predicting the contours (compressed pixel indices) of the lanes which are further processed to filter out false positives. Non-max suppression is employed to eliminate overlapping detections, ensuring that each detected lane is unique and accurately represented. A binary mask that highlights the lane lines from the segmentation predictions and converts them to a mask format. These masks are then used to find contours, which represent the lane. These contours are filtered based on the area to remove small, irrelevant detections, leaving only the significant lane markings. In figure 1, the lane lines are highlighted with blue color. These pixel indices are then mapped on the blender world coordinate frame using ZoeDepth [2]. The depth estimation for this scene can be seen as a grayscale image in 2. Using pixel locations of each pixel, camera intrinsics, and depth estimation for that pixel, we can use simple trigonometry to find the projection of each pixel in Blender world coordinate frames. We reject projections that lie 10cm above the the ground as we know that lane markings need to be on the ground. We visualize the 3D projected points in 3. For each of the lane lines, we fit a 2nd-degree polynomial. We break the line into 3 meter-long segments and sample the centroid of each segment. This centroid is the origin of the rectangle spawned in Blender. To get the direction of the rectangle, we also look at the derivative of the centroid.

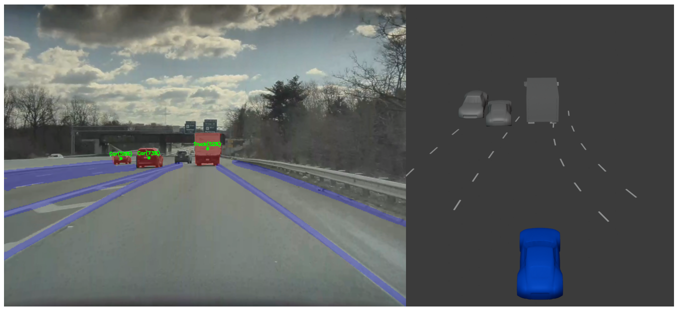

Fig. 1. Side-By-Side Annotation and Rendered Output for the Lane Detection Task (Scene 1, Frame 368)

B. Vehicle Detection

We used YOLOv8-seg for detecting the contour masks of the vehicles. Detected objects are filtered to include only relevant categories such as cars, trucks, motorcycles, and Bicycles. Detected vehicles are annotated on the original images, by drawing contours around the vehicles, marking centroids, and adding text labels indicating the object class and confidence levels. Each of the vehicles are then projected on the Blender world coordinates using depth estimation similar to as described in II.A. The difference in vehicle projection is as follows. or cars, 1.5 meters is added to the depth estimation of the centroid of the car in the image because this would give a better depth estimation of the 3D centroid of the car. For trucks, this is 2 meters. For motorcycles and bicycles, this is 0.5 meters.

For each of the detected vehicles, we also run 3D Pose estimation as described in [] and YOLO3D [refs]. However,

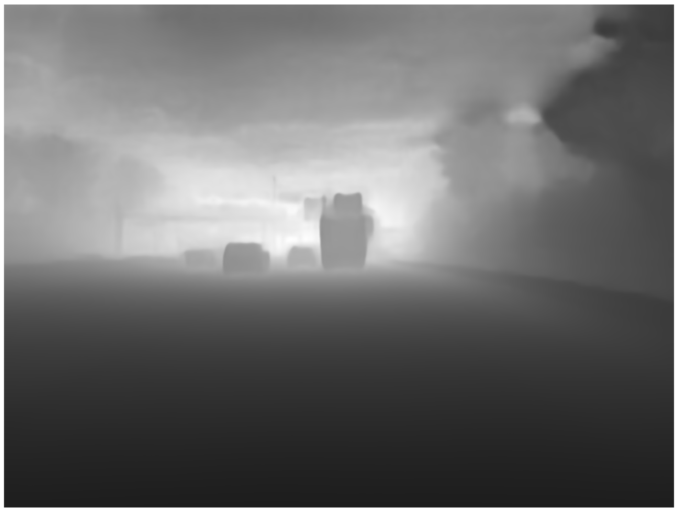

Fig. 2. The Depth Estimation Output for the Frame Used for 1

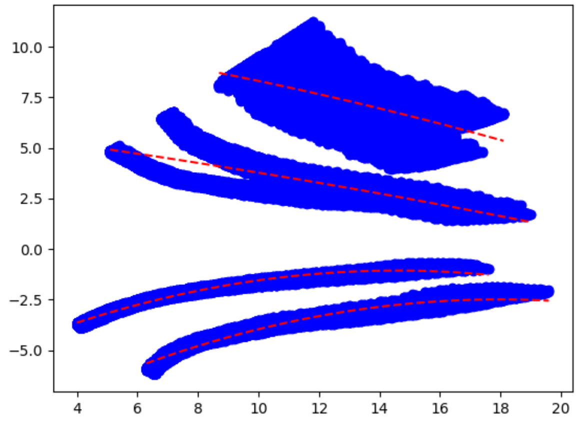

Fig. 3. 3D Projection of Lane Lines in Blender World Coordinates

since we are only interested in the Z-axis rotation, we don't visualize the bounding box.

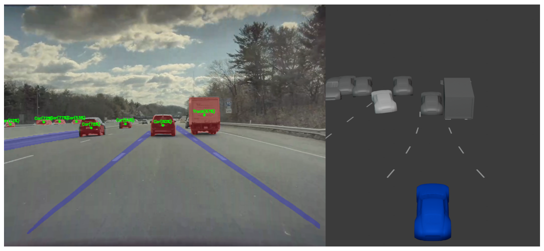

Fig. 4. Visualization Output Showing Detection of Cars and Trucks

C. Pedestrian Detection

In this task, we use the work of [4] to detect pedestrians and their pose. This work relies on YOLOv5 for person detection and another model to detect the human pose. We use the

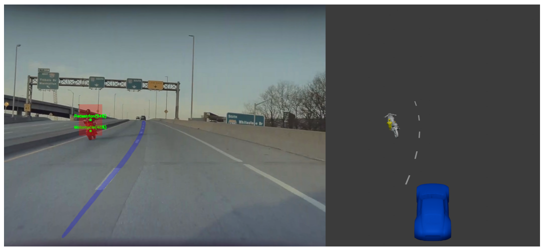

Fig. 5. Visualization Output Showing Detection of Motorcycles

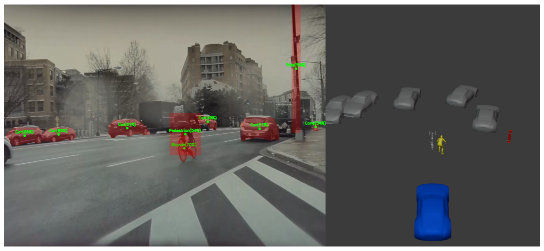

Fig. 6. Visualization Output Showing Detection of Bicycles

bounding box information and the generated .obj files to render pedestrians.

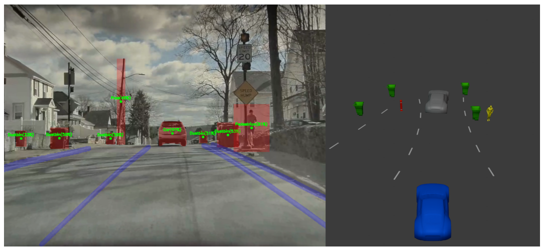

Fig. 7. Visualization Output Showing Detection of Pedestrian

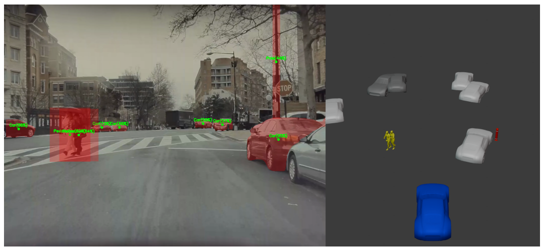

Fig. 8. Visualization Output Showing Detection of Another Pedestrian

D. Traffic Signs Detection and Classification

The YOLOv8 object detections include traffic lights, however, color is not detected. Initially we performed HSV thresholding without position logic, which gave many edge cases. Therefore we added position logic also to it. We crop the region of interest from the bounding boxes and keep only the center 50% of the cropped ROI to make sure the pixels outside the traffic light don't affect the average brightness calculations. We perform color space conversion from RGB to HSV for the color segmentation task. The value channel of the HSV image which represents brightness, is analyzed to identify the illuminated section of the traffic lights. The resized traffic light is divided vertically into three equal sections, corresponding to the typical arrangement of traffic lights(red at the top, yellow in the middle, and green at the bottom). The brightness levels within each section are summed up to determine the most illuminated section. The section with the highest brightness sum is presumed to be the illuminated part of the traffic light. Based on its position(top, middle, or bottom), the traffic light is classified as red, yellow, or green, respectively.

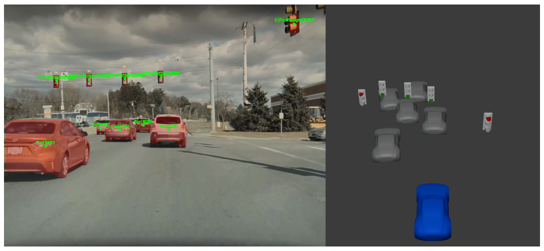
Fig. 9. Visualization Output Showing Detection of Traffic Lights and Their Color

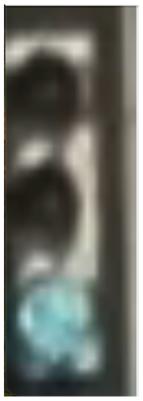
Fig. 10. Cropped Traffic Light

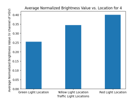
Fig. 11. Average Normalized Brightness Traffic Light Color Locations

E. Road Signs

For stop sign detection we use YOLOv8-seg model again and render in blender based on the texture to spawn a new object in the blender world.

F. Challenges

The curvature of the polynomial is dynamic and becomes high for straight lines as well. The traffic light color detection was done using HSV thresholding, however during night times and flaring environments adaptive thresholding didn't work out

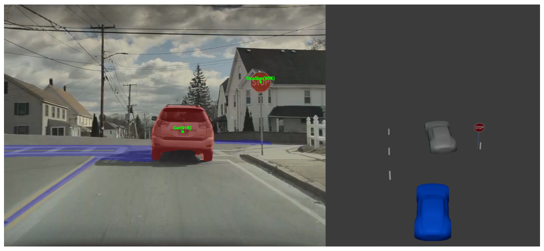
Fig. 12. Stop Sign Detection

well. This approach was changed to brighntess estimate based on luminance and position estimates.

III. II. Phase 2: Advanced Features

A. Vehicles

3D poses of vehicles are done by integrating the Yolo3D model into our existing Yolov8 detection framework. For each detected vehicle, orientation and dimensions are estimated using a *ResNet18*-based regression model. This involves cropping the detected vehicle from the image, resizing it, and normalizing it before feeding it into the network. For each vehicle, the angle of orientation(**ry**) is calculated by combining the model's orientation output with the vehicle's centroid position relative to the camera's perspective. This angle is crucial for understanding the vehicle's direction relative to the camera's viewpoint.

B. Road Signs

1) Speed Sign: We created a traditional approach system that uses optical character recognition (OCR) and computer vision techniques to identify speed limit signs in frames. It scans images, uses HSV color space filtering to extract bright areas that might be speed signs, and recognizes contours that roughly match the dimensions and design of speed limit signs. When a contour meets these requirements, the enclosed region is examined and Pytesseract is used for optical character recognition (OCR) to extract numeric text. This text is then verified to correspond with standard speed limit values.

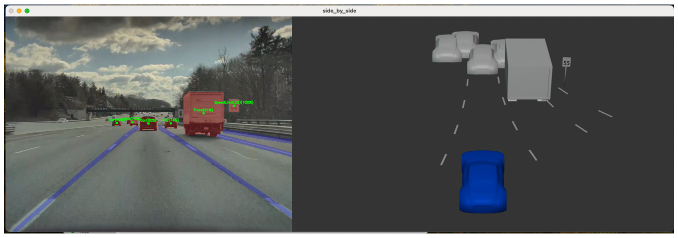

Fig. 13. Speed Limit Detection

2) Arrow detection: To highlight bright areas suggesting arrows, color space conversion, and thresholding are two image processing techniques that are used to isolate possible arrow regions in the driveable area. These highlighted regions' contours are then retrieved, and their size is used to filter out unlikely possibilities. The orientation of each arrow is determined for each contour that remains by converting 2D image coordinates into 3D world coordinates using depth information and camera calibration data. This orientation denotes the arrow's recommended direction, such as turning or moving straight ahead.

To get direction, we simply fit a line through the arrow. However, this is not enough to get the full direction as we won't be able to differentiate between ahead and towards. For this, we compare the bounding box center and mass centroid of the arrow.

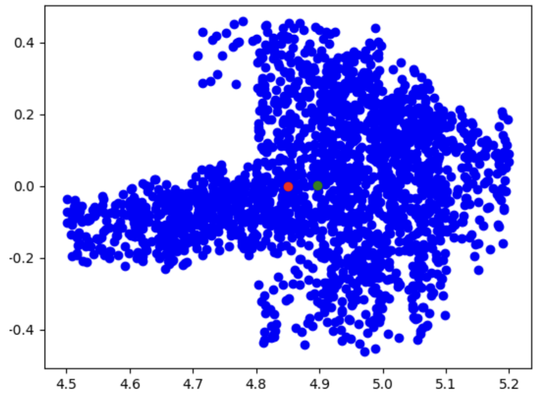

Fig. 14. Contours of Detected ground arrow

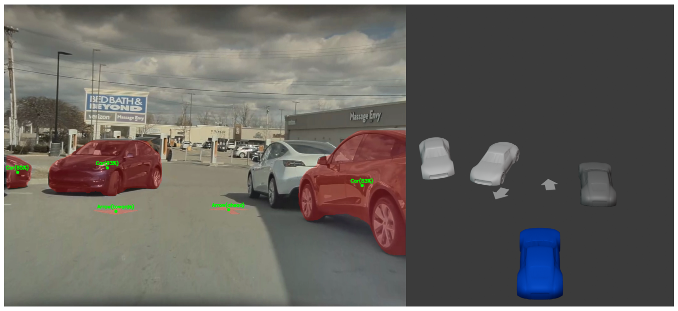

Fig. 15. Ground Arrow Detection

C. Other Objects

Other objects like road traffic cones, dustbins, and poles are detected by the Detic model. We save the contour information of each object based on the custom vocabulary dictionary like cones, poles, dustbins and parse their confidence scores.

D. Challenges

Detecting the arrow on traffic signs was challenging work, using a custom-trained network for this task was not robust as the network was biased and gave only left signs. Applying traditional thresholding was erroneous due to environmental glare and different lighting conditions. We initially used Zoe depth[2] for getting the metric depth, however, scaling had to be fine-tuned which changed in different scenes. The pedestrian mesh generation model used was HybrIK, however it identified only 1 pedestrian per frame. Idea-OSX require a

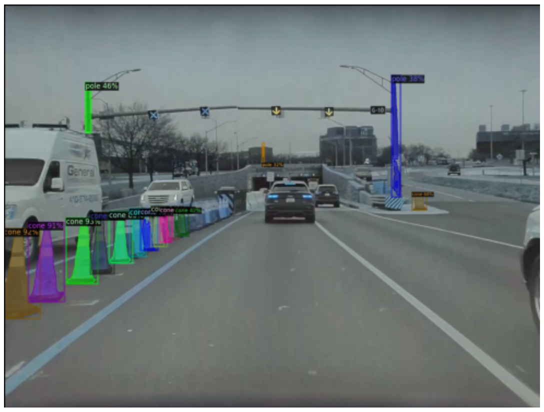

Fig. 16. Sample Detic Detection for other objects

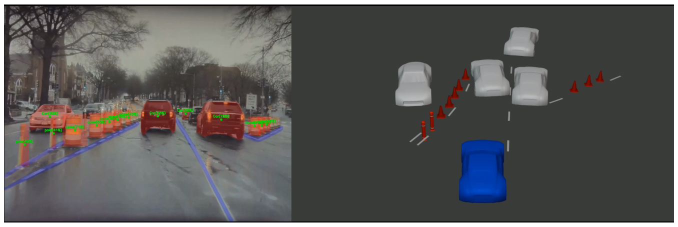

Fig. 17. Cone and Poles Detection

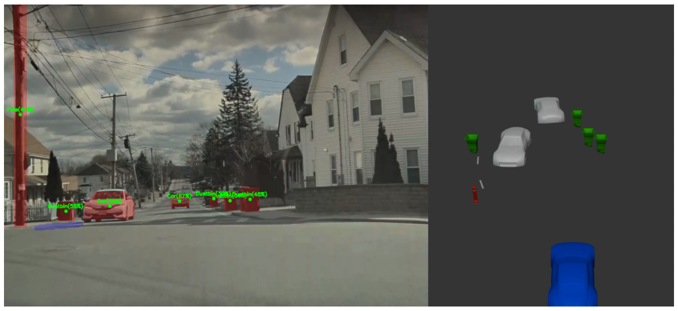

Fig. 18. Dustbin Detection

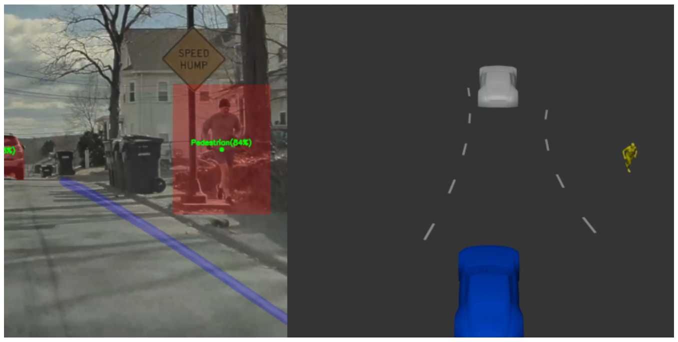

Fig. 19. Pedestrian Pose estimation and Speed Limit Edge Case

bunch of human model files and locating the complete human models can be found here[3]. Speed Limit detection was not robust, as seen in Figure 9. Pedestrian is detected but speed limit sign detection is missed. This can be made more reliable by incorporating a better region of interest estimation like a trained model on LISA dataset than traditional approach.

IV. III. Phase 3: Bells and Whistles

A. Brake lights

The contours of cars are recognized using YOLOv8-seg model. We determine places inside the car's shape that have been identified as probable tail or brake lights based on DETIC model predictions. This is accomplished by comparing the luminance (brightness) of specific spots to the total luminance of the automobile area; a substantial difference indicates that a light is probably on.

We calculate the luminance of each light area by converting the relevant part of the image to a YCrCb color space, which separates the luminance (brightness) and chroma (color) components, allowing for a simple computation of brightness. If the brightness of a light zone surpasses a predetermined threshold when compared to the overall brightness of the vehicle, the light is considered ”on”. The position of the light (left or right side of the automobile) also influences the car's status: both lights on may signal that the car is stopped, whilst one light may indicate a turn.

B. Parked and Moving vehicles

We use flowmap from Optical flow model RAFT[4]. The dark gray vehicles are stationary vehicles and the white vehicles are moving as detected from the model.

C. Challenges

Thresholding luminance values didn't perform well during night times. We tried to apply absolute(thresholding luminance to 550) and relative luminance(tail light luminance greater than car luminance) logic however that affected our daytime brake detections. Therefore, more robust approach is needed, probably vehicle velocity estimation from consecutive frames would help this approach.

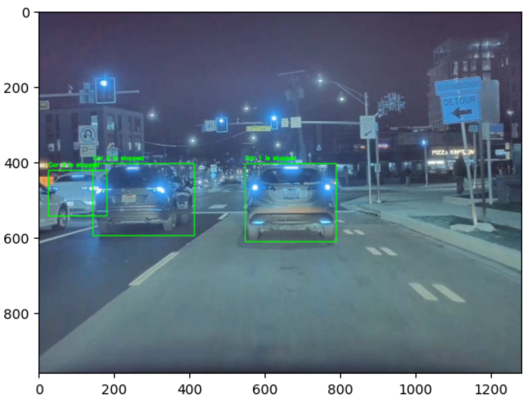

Fig. 20. Brakes applied estimation sample output

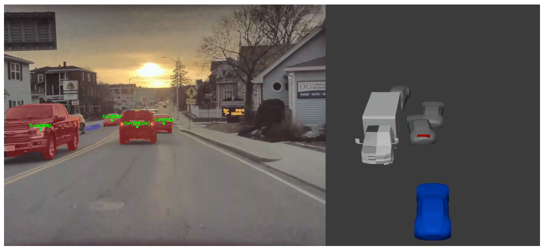

Fig. 21. Brakes applied estimation sample output

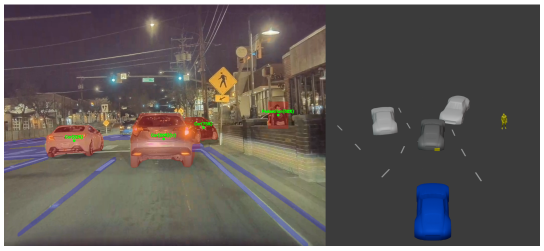

Fig. 22. Turn Signal applied estimation sample output

Fig. 23. Grayscale Flow for Parked and Moving Vehicle Estimation

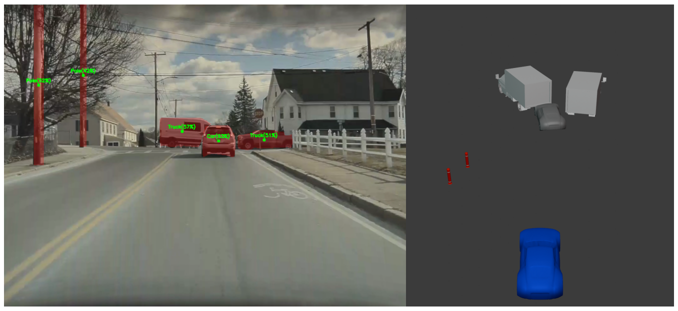

Fig. 24. Parked and Moving Vehicle Estimation

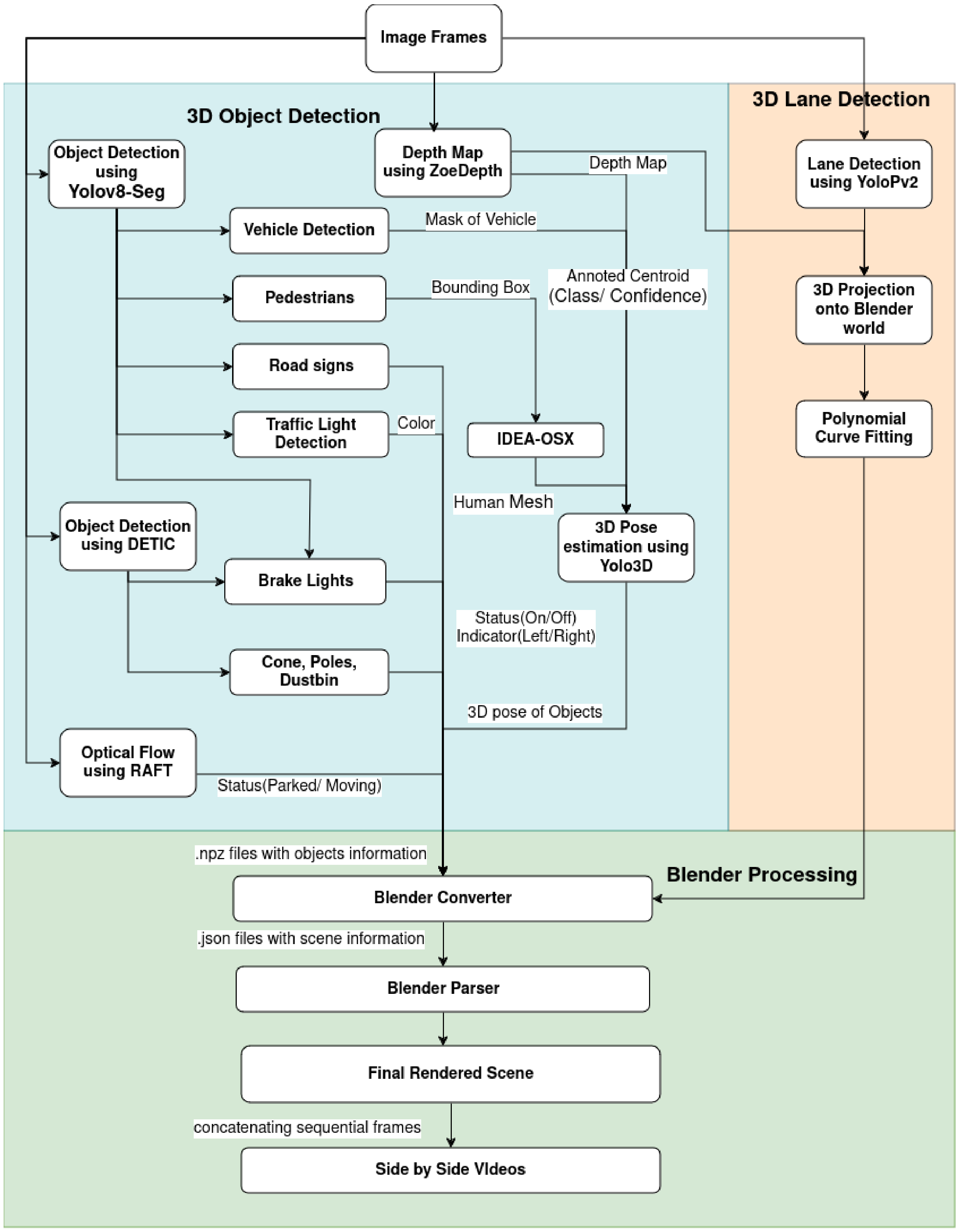

Fig. 25. Driving scene semantic rendering pipeline

V. Conclusion

References

[1] M. Diaz-Zapata, Ö. Erkent and C. Laugier, ”YOLO-based Panoptic Segmentation Network,” 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 2021, pp. 1230-1234, doi: 10.1109/COMPSAC51774.2021.00170. keywords: Measurement;Navigation;Conferences;Semantics;Object detection;Real-time systems;Software;Panoptic Segmentation;YOLOv3;Autonomous Vehicles,

[2]

[3] https://github.com/isl-org/ZoeDepth

[4] https://github.com/IDEA-Research/OSX human-model-files: https://wpi0-my.sharepoint.com/:u:/g/personal/hbhat_wpi_edu/ERMS0sCw-gpGubgkTTUXBkQBir4lOC2bOU9XyVZnXO4Okw?e=Q03Spb

[5] Teed, Zachary and Jia Deng. “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow.” European Conference on Computer Vision (2020).