P3 - Einstein Vision

Mihir Deshmukh
Robotics Engineering
Worcester Polytechnic Institute
Email: mpdeshmukh@wpi.edu

Ashwin Disa
Robotics Engineering
Worcester Polytechnic Institute
Email: amdisa@wpi.edu

Using 1 LATE day

***Abstract*—In this project, we develop an intuitive, vision-based dashboard aimed at augmenting human drivers' experience on the road. Inspired by Tesla's dashboard, this work integrates deep learning techniques to tackle the complexities inherent in autonomous driving. Central to this project is the creation of a 3D representation of the surrounding environment through the detection of vehicles, road signs, lane markings, pedestrians, and other objects. By leveraging techniques such as Object Detection, Depth Mapping, and Pose Estimation, a seamless fusion of data and visualization is achieved.**

I. Pipeline overview

We received undistorted videos for 13 different scenes and 4 angles collected from a Tesla Model S. We opted to utilize the front camera view for object visualization. For processing, we select one image frame every 10 frames, run inference on it, and save the information such as type of object, its position, rotation, etc. for each frame in a scene we store using a particular JSON format to facilitate rendering of the detected objects. The same JSON file is read in Blender using the scripting box in Blender and based on the information in the JSON we use the provided blender models as well as certain custom ones for different objects such as vehicles, pedestrians, traffic lights, speed limit signs, etc.

II. Checkpoint 1: Basic features

In the first phase, we implement basic features that are absolutely essential for a self-driving car. Includes the following: different types of lanes, vehicles (without classification), pedestrians (without pose), Traffic lights, and stop signs.

A. Lane detection

Lane detection holds significant importance for autonomous vehicles, particularly for their navigation. Initially, we implemented CLRerNet [1] but unfortunately, it was unable to detect the type of the lane. We employed a mask RCNN model [2] trained on a custom lane detection dataset to detect various types of lanes such as solid, dotted, divider line, and double line. Subsequently, we utilized this information to accurately plot the lanes in Blender. We extract the points of interest from the mask using the following algorithm:

- Take the binary mask output from the model.
- Extract nonzero coordinates (ys, xs) from a binary mask, returning an empty array if no points are found.
- Generate a set of evenly spaced target y-values, y_new, between the minimum and maximum of ys.
- For each y in y_new:
  1) Find ys indices close to y; if none, interpolate x from ys and xs.
  2) If indices are found, compute the mean of corresponding xs.
- Pair the computed or interpolated x values with y_new to form new points.
- Return these points as an integer array.

We sample six such evenly distributed points on the mask and store them. To translate the predicted lanes i.e. these points into the 3D metric space, we employed monocular depth estimation on the same image to estimate the depth of each pixel within the lane. The resultant output provided a list of 3D points in the world space, with their height assumed to be at ground level (Z=0). These points were then used to generate a bezier curve which fit a smooth curve between those points and visualized within Blender. Additionally, the lane type information was utilized to color code each segment accordingly. The model was also able to detect road signs i.e. the arrows on the ground which we will talk about later.

The lane detection worked significantly better on highway scenes than on city scenes where the lane markings were faded and not marked properly.

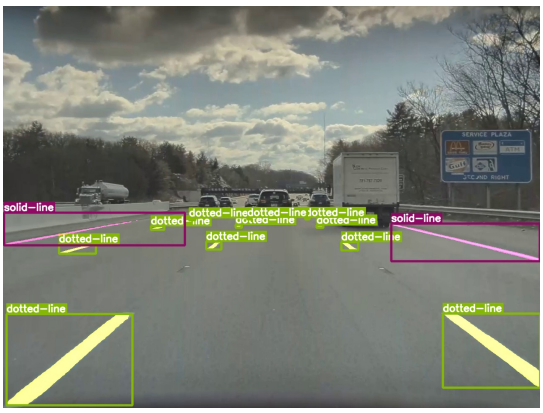

Fig. 1. Lane detection.

B. Vehicle, pedestrian, stop sign detection

For the first phase, we are required to detect the cars and not classify them. Initially, we used YOLOv9 [3] for all

detections. YOLOv9 is robust and easy to infer directly from an image input. The output is the bounding box, mask, and the confidence score of each detected object. The depth map of the same image was found using ZoeDepth. Using the object's bounding box, we estimated the pixel coordinates and mapped them to the depth image. ZoeDepth [4] was initially used to estimate the metric depth. Using the camera intrinsics, pixel values, and metric depth, we estimated the 3D point of the object and visualized it in Blender. A rotation matrix was employed to rotate the 3D points with respect to the camera frame. The Z-axis (depth) is perpendicular to the image plane. The equation for the pixel to the 3D point is given below.

$$x = (u - c_x) * z/f_x$$
$$y = (v - c_y) * z/f_y$$

where f_x, f_y are the focal lengths and equal to $1594.7, 1607.7$ mm. The principal points c_x, c_y are $655.2961, 414.3627$ pixels respectively. z is the depth and u, v are the pixel points of the object in the image frame. The output x, y are the points in the 3D world frame. We later found a better and more recent depth map estimation model, Marigold [5] which gives a more robust output with accurate depth information compared to Zeodepth.

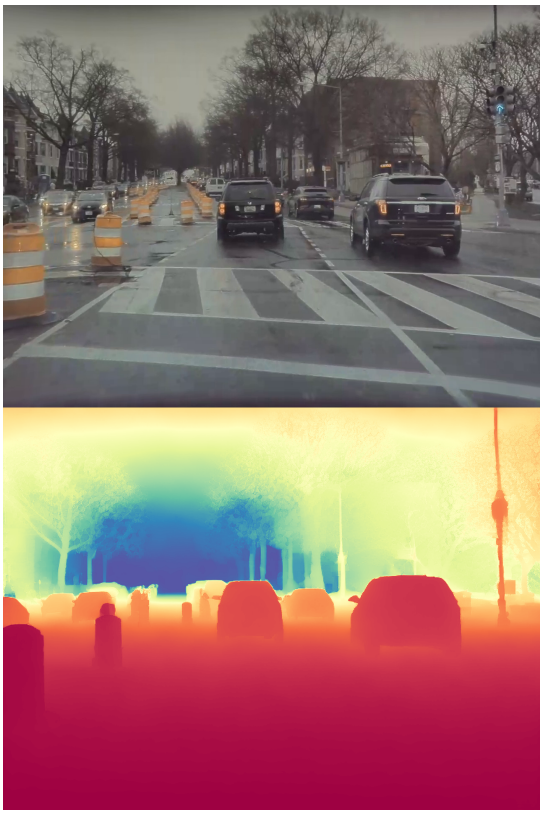

Fig. 2. Colored depth image using Marigold framework

C. Traffic light detection

YOLOv9 was able to detect the traffic signal but was not able to identify its color. To do so, we found a model [6] which had pre-trained weights and built over the YOLOv3 framework. This model was able to classify the green light as "go", the red light as "stop", the yellow light as "warning", the left arrow as "goLeft" and so on. However, the detection isn't very robust as it seems to miss detections from far away. We would need to train on a larger dataset for more robust results.

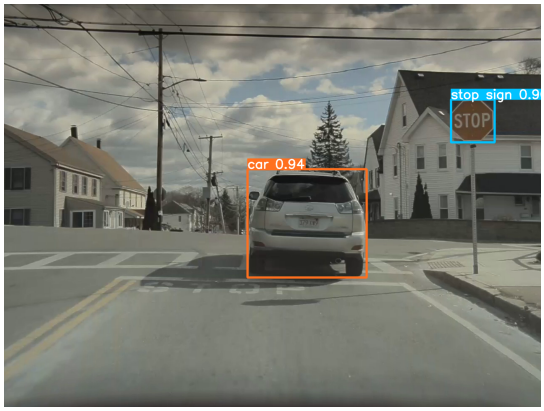

Fig. 3. YOLOv9 output with the car, stop sign, scene 5 intersection

Fig. 4. YOLOv3 traffic light detection

III. Checkpoint 2: Advanced features

In this phase, we built over the previous work by enhancing and adding more features.

A. Vehicle classification

In this phase, we replaced the YOLOv9 object detection framework with Facebook's DETIC [7] to do instance segmentation. DETIC gives the existing COCO dataset classes additionally also having the lvis classes. It also works with a custom dictionary which we use to identify the car sub-types such as sedan, SUV, pickup truck, hatchback, and truck using custom vocabulary. We do essentially a non-max suppression to get rid of multiple masks on the vehicles. The other classes were identified using the standard predictor including motorcycles and bicycles. This would not be possible using the previous framework as it was not trained using these labels as

well as doesn't allow for a custom vocabulary. The car subtype classifications can be seen in Fig. 5. As we are using a custom vocabulary, the confidence of the predictions is on the lower side as it uses CLIP embeddings as input. Hence we only opt to identify car sub-types using the custom vocabulary while the other objects like stop signs, cones, barrels, etc from the standard predictor.

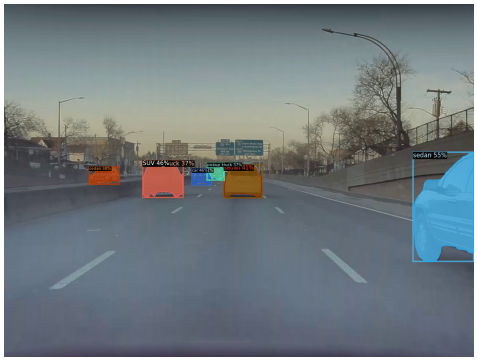

Fig. 5. Detic Car Sub type detection

B. Vehicle pose estimation

We also had to estimate the orientation of the vehicles for this phase. We use the implementation of YOLO3D [8] trained on the KITTI dataset. We pass the bounding boxes from DETIC to the regressor which predicts the 3D bounding box. We use the yaw from this to spawn the cars with the corresponding orientations. The model used the image corp from DETIC i.e. the 2D bounding box from DETIC predictions, to estimate the 3D location and back project onto the image. The results were pretty accurate for all vehicle subtypes except the trucks in the scene which had an offset in the orientation. The possible reason could be the smaller number of truck images in the dataset compared to other vehicles. However, it should be noted that the yaw from this is not very accurate when the car is very close or is occluded. A more advanced network might be utilized for better results. Fig. 6 shows a typical result from YOLO3D where the Blue X denoted the front of the car. Fig. 7 shows a failure case where the yaw is not accurate when the cars are too close. This is apparent in scenes with traffic and the yaw artifacts can be seen in the rendered videos.

C. Road signs

Along with the stop sign we also had to detect road signs on the ground such as arrows and speed limit signs along with the numbers. We used the pre-trained YOLOv8 weights from the GLARE dataset paper [9] for identifying various signs. The speed limit detection was found to be inaccurate with multiple failures in detecting the actual limit. The arrows on the ground were detected by the same lane detection mechanism used for detecting the lane and its types. The output is a list of points of the contour boundary of the arrow and a bezier curve is fit along it. The contour detection and its' estimated depth are

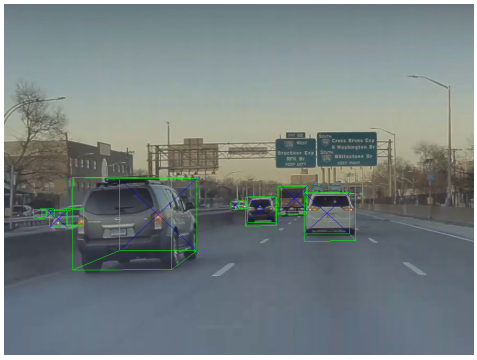

Fig. 6. 3D Bounding box from YOLO 3D

Fig. 7. 3D Bounding box Failure cases from YOLO 3D

reliable which made the results robust. The model was also able to detect the cycling lane markings on the ground in some instances. Fig. 9 and 8 show detections of some common road signs.

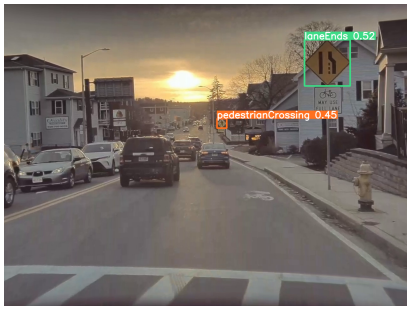

Fig. 8. Road signs detection using YoloV8 (Pretrained Glare Dataset)

D. Miscellaneous objects

Apart from the trivial objects, there are plenty of objects such as dustbins, traffic cones, and cylinders present in the surroundings which we aim to detect. DETIC was able to handle the detection of these objects and was found to be pretty robust. The blender models for all such objects were already provided. Using the depth information, each object was seamlessly spawned in the environment.

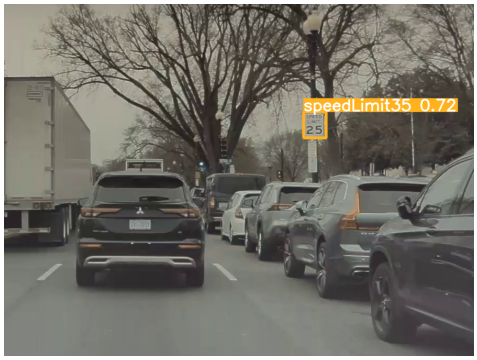

Fig. 9. Speed limit detection

Fig. 10. Cones

Fig. 11. Trashcan

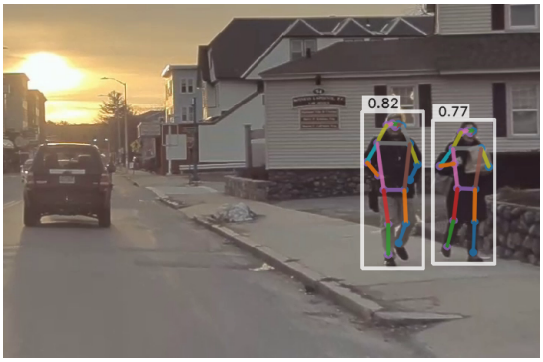

Fig. 12. Human mesh using yolo nas pose.

E. Pedestrian pose estimation

We explored two approaches to estimate human pose from detected pedestrians. Initially, we aimed to leverage the yolo_nas_pose [10] framework, which provides key point coordinates representing different body parts. The intention was to map these key points onto an armature within Blender to animate a human mesh. However, grappling with the complexities of determining interlink angles for the armature proved challenging, prompting us to seek an alternative solution.

Subsequently, we turned to the OSX [11] framework, offering a different methodology. By inputting monocular images of human figures, OSX generates 3D mesh representations encapsulated in *.obj* files. This streamlined our workflow by providing readily usable mesh data for integration into Blender. To ensure precise positioning within our Blender environment, we integrated depth and pixel information from the input images to estimate the 3D world coordinates of the mesh vertices. This approach facilitated the seamless integration of human poses into our Blender scenes, enhancing the overall efficiency and effectiveness of our project workflow.

IV. Checkpoint 3: Bells and Whistles

A. Brake light and Indicators

The analysis of vehicle light status, specifically brake lights and indicators, employs the YCbCr color space due to its effective separation of luminance from chrominance components. This separation is advantageous for identifying specific color features under various lighting conditions. The following steps outline the methodology:

1) Take the input as a cropped image from the DETIC mask.
2) Initially, the image undergoes Gaussian Blur filtering to reduce noise.
3) The image is converted to the YCbCr color space, emphasizing the Cr channel to highlight red hues.
4) A dynamic thresholding technique is applied, leveraging the mean and standard deviation of the Cr channel, to isolate significant red areas. This threshold is adjusted to optimize the differentiation of red areas from the background.
5) The binary mask is analyzed to quantify red areas in the left and right sections of the image. This quantification aids in determining the light status based on the distribution of red areas.
6) A decision margin, calculated as a percentage of the total red areas, is introduced to differentiate between the statuses of the left/right indicators and brake lights.
7) The analysis concludes with the identification of the vehicle light status, offering determinations of whether the left or right indicator is on, the brake light is on, or the brake light is off.

Some drawbacks that we were able to see from the results are during nighttime as all the lights are on, it sometimes misses the left and right indicators as the brightness variance is low in some cases. This can be made more robust if we use the taillight masks instead of the whole car. Also to account for left and right from the cars from opposite directions, we can include the flow directions calculated in the next section.

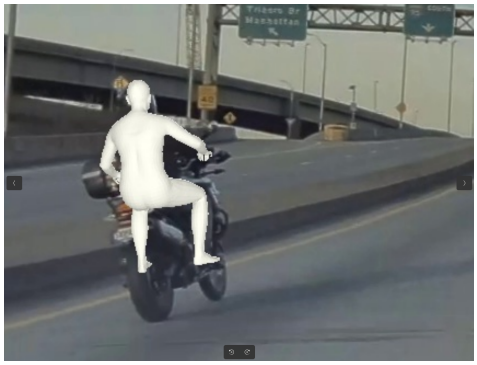

Fig. 13. Human mesh using OSX.

Fig. 14. Brake lights detection using YCbCr thresholding

B. Parked and moving vehicles

To distinguish between parked and moving vehicles, we utilized optical flow analysis with the RAFT algorithm [12], which provides monocular optical flow images indicating relative flow between pixels. Higher flow rates are represented by intensified colors in the flow images. Initially, we attempted to use the Sampson distance to mask moving vehicles based on their higher flow rates. However, we encountered a challenge where vehicles in front, moving relatively slower compared to our car, were mistakenly categorized as parked due to their low flow rates. To address this issue, we devised a multi-step approach. Firstly, we evaluated the net flow within each bounding box provided by DETIC and compared it with the flow from neighboring regions. If the difference in flow was relatively low, indicating minimal movement of the object relative to the scene, and if the net flow of the mask itself fell within a specified range (indicating that cars moving in the same direction were below a minimum threshold), we categorized the vehicle as parked. Additionally, to account for our car's motion, we employed flow subtraction based on a Gaussian mask, with the variance of the mask proportional to the variance of flow in the image. This approach allowed us to better distinguish between parked and moving vehicles. The method can be summarised as follows:

1) Apply spatial weighting to the optical flow field from RAFT, focusing on vector proximity to the image center and intensity.
2) Assess optical flow within bounding boxes of detected vehicles, comparing it to adjacent flow.
3) Classify vehicles based on flow magnitude differences:
   - Vehicles with minimal relative movement compared to surroundings are classified as *parked*.
   - Vehicles with flow magnitude significantly higher than the surroundings are classified as *moving*.
4) Enhancements can be done by incorporating absolute movement data for a more stable basis in movement analysis and reducing false positives.

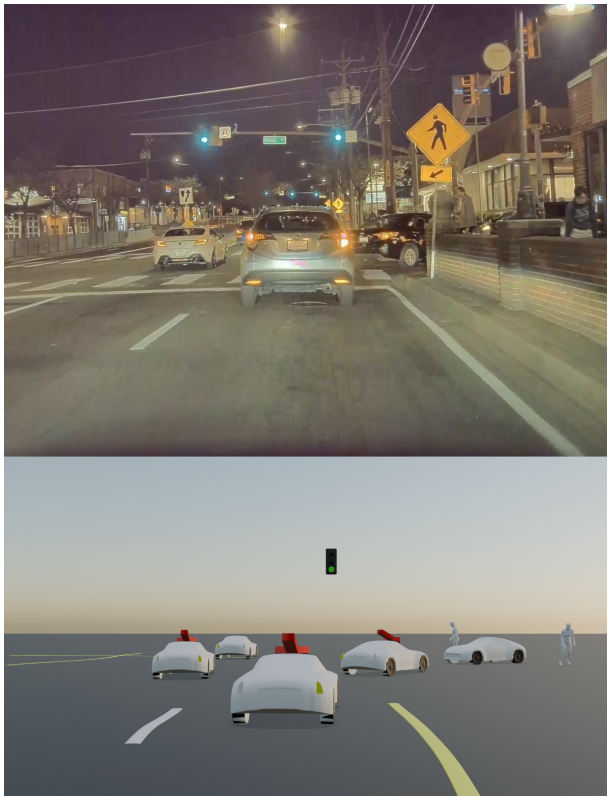

Fig. 15. Right indicator example.

The flow output of the network is relative, hence the weighted flow negation based on the variance of the image and flow improves the results but still it is not very robust. A better flow negation method would include the odometry data of the car i.e. the camera. This will result in a much more robust output.

V. Extra Credit: Speed bumps

For speed bump detection, we can see in the video sequences, there is a sign associated with it. We use the [13] dataset from Roboflow to train a YOLOv8 model to detect speed bump signs as seen in Fig.18. Here, we assume the speed bump is just beside the sign, which is the case for scene 9 as seen in FigṪhis works well for scene 9 but this approach fails in scene 5 as the speed bump is ahead of the sign and we need to identify the speed bump in another way to accommodate these changes. We tried to look at the depth map to see if we got variations on the road, but it wasn't a viable approach as the monocular depth map didn't capture these small details. We also thought about identifying the sign on the road i.e.

Fig. 16. Optical flow.

Fig. 17. Parked and moving vehicles detections from optical flow.

the triangle on the speed bump Fig.18 which might be used for determining the speed bump, but it is also not consistent across all streets. The output for the speed bump rendered in the video can be seen in Fig.19.

The final rendered images can be seen in Fig.20 and Fig.26 The overall pipeline can be visualized as follows 27:

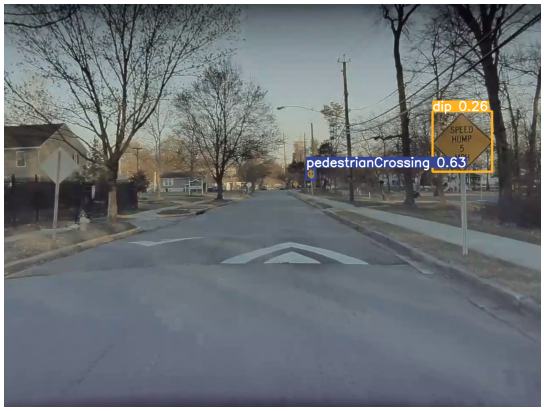

Fig. 18. Speed Bump sign identification: Scene 9

Fig. 19. Speed Bump spawned in Scene 5

REFERENCES

[1] https://github.com/hirotomusiker/clrernet.
[2] https://debuggercafe.com/lane-detection-using-mask-rcnn/.
[3] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9: Learning what you want to learn using programmable gradient information, 2024.
[4] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth, 2023.
[5] Bingxin Ke, Anton Obukhov, Shengyu Huang, Nando Metzger, Rodrigo Caye Daudt, and Konrad Schindler. Repurposing diffusion-based image generators for monocular depth estimation, 2024.
[6] https://github.com/sovit-123/traffic-light-detection-using-yolov3?tab=readme-ov-file.
[7] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. Detecting twenty-thousand classes using image-level supervision, 2022.
[8] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry, 2017.
[9] Nicholas Gray, Megan Moraes, Jiang Bian, Alex Wang, Allen Tian, Kurt Wilson, Yan Huang, Haoyi Xiong, and Zhishan Guo. Glare: A dataset

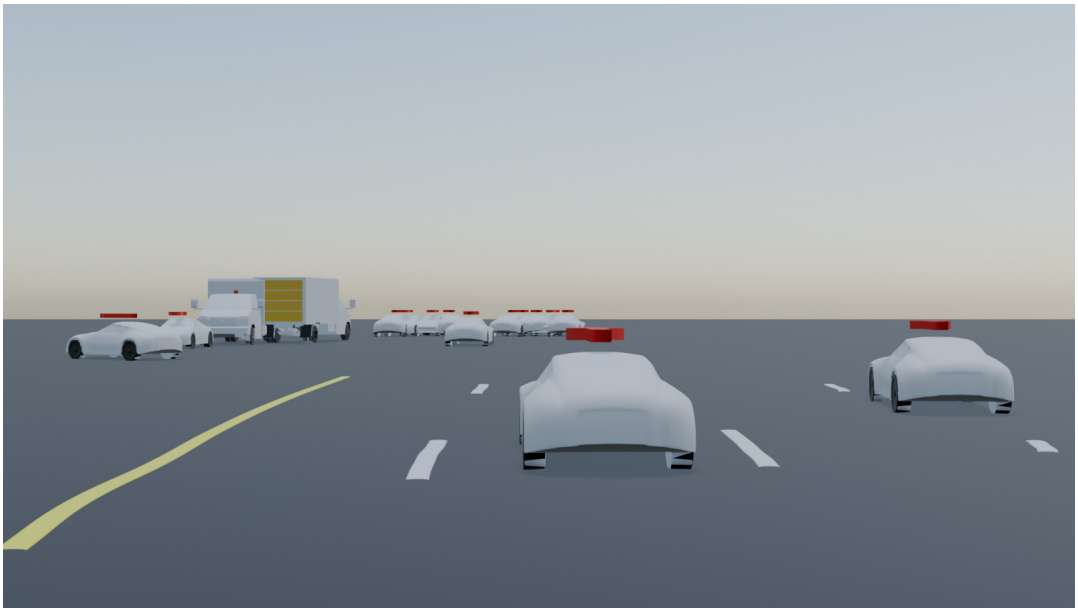

Fig. 20. Final Rendered Image sample - 1

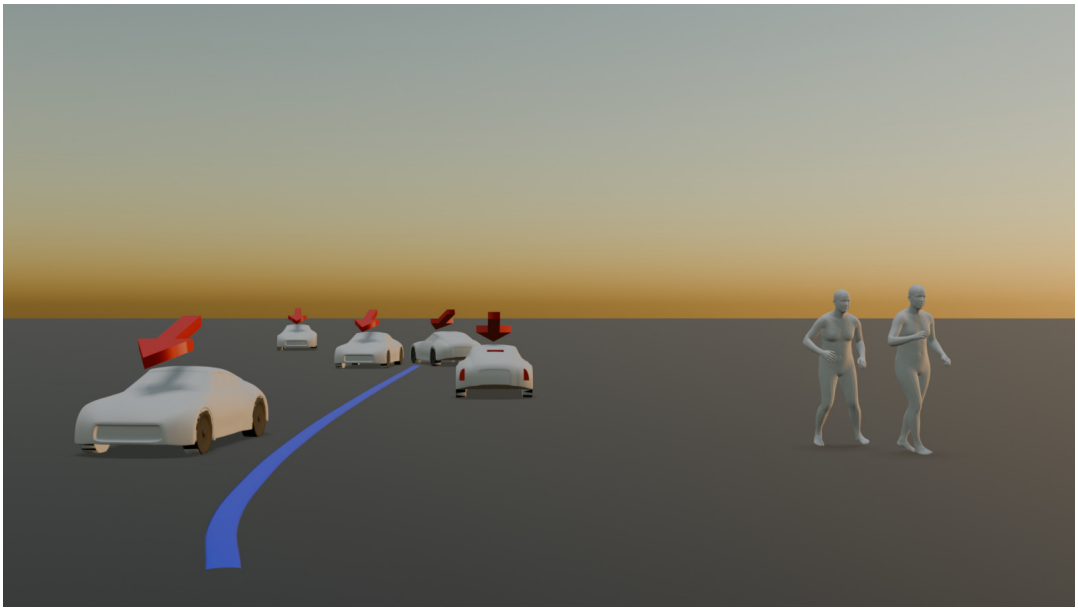

Fig. 21. Final Rendered Image sample - 2

for traffic sign detection in sun glare. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):12323–12330, 2023.

[10] https://github.com/deci-ai/super-gradients/blob/master/yolonas-pose.md.

[11] Jing Lin, Ailing Zeng, Haoqian Wang, Lei Zhang, and Yu Li. One-stage 3d whole-body mesh recovery with component aware transformer, 2023.

[12] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. *CoRR*, abs/2003.12039, 2020.

[13] https://universe.roboflow.com/dakota-smith/lisa-road-signs.

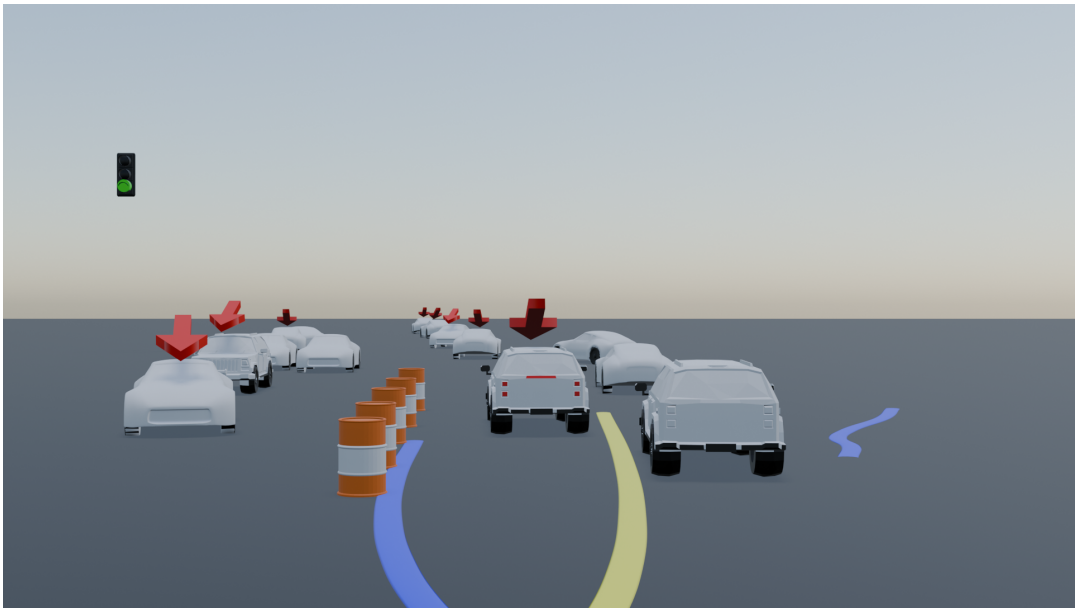

Fig. 22. Final Rendered Image sample - 3

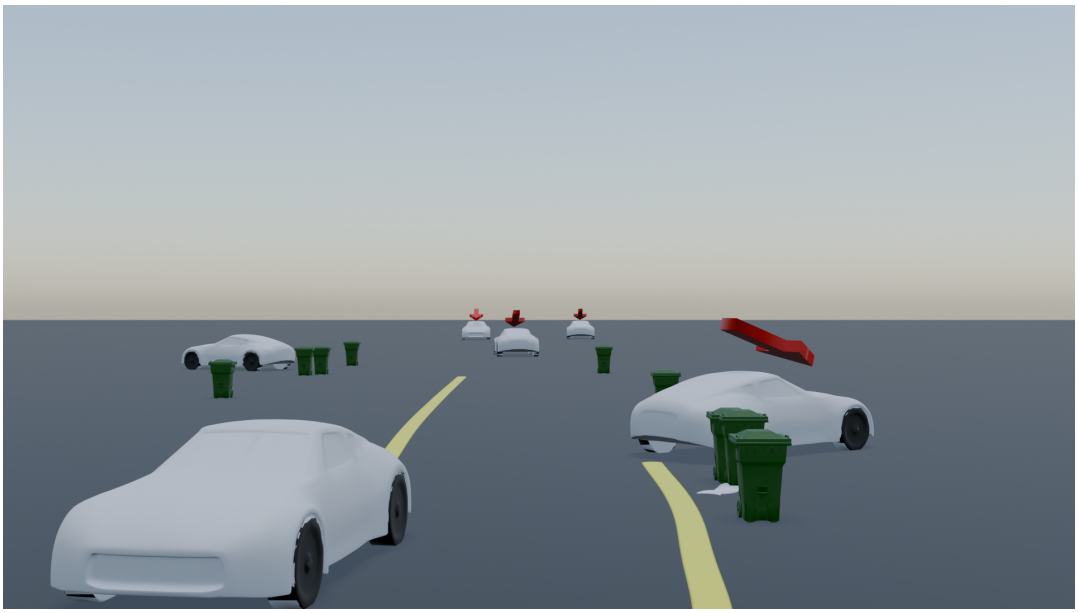

Fig. 23. Final Rendered Image sample - 4

Fig. 24. Final Rendered Image sample - 5

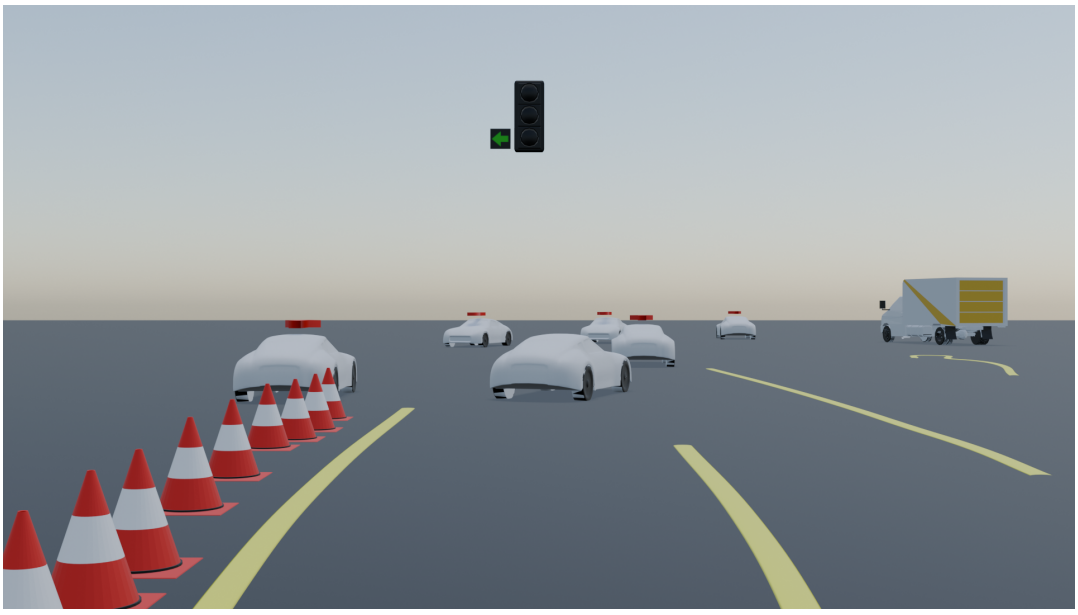

Fig. 25. Final Rendered Image sample - 6

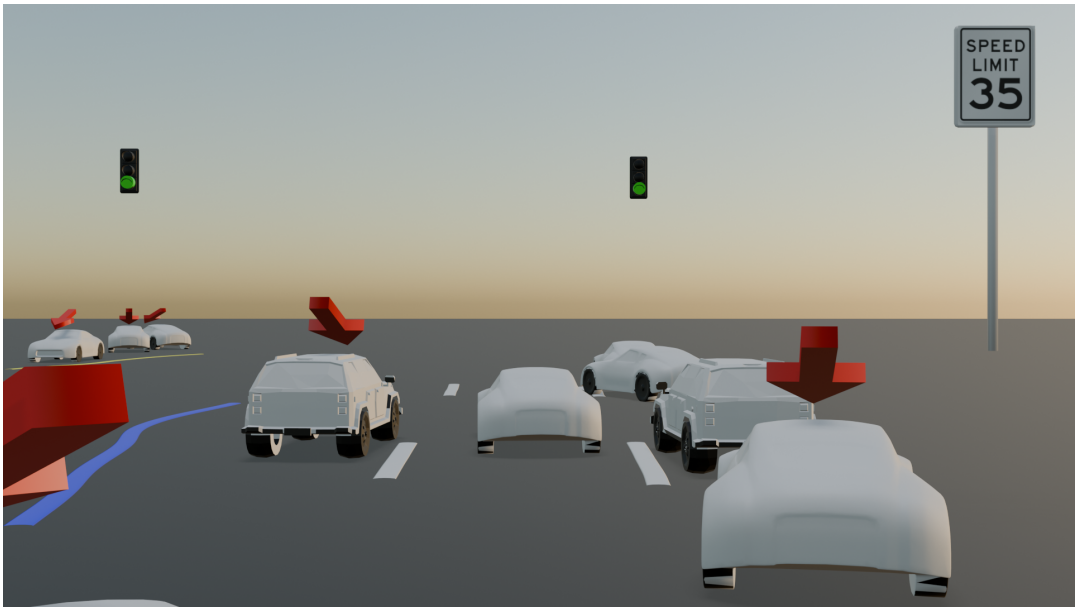

Fig. 26. Final Rendered Image sample - 7

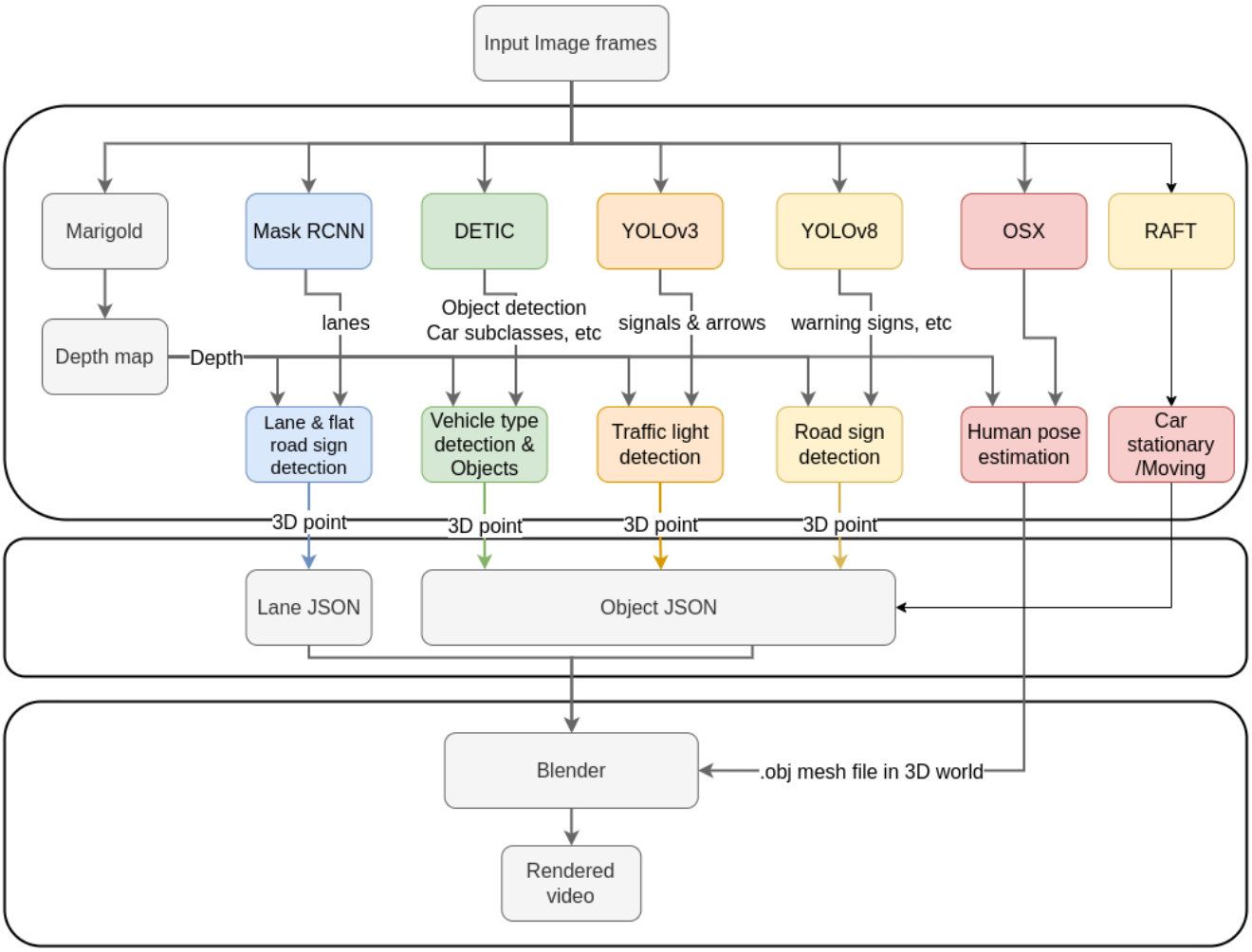

Fig. 27. Overall system pipeline