Project 3 - EinsteinVision

Harmeet Dhillon
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA - 01609
Email: hdhillon@wpi.edu
Using 4 Late Days

Rohan Walia
Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA - 01609
Email: rwalia@wpi.edu
Using 4 Late Days

***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. Overview

We worked with undistorted videos captured from 13 different scenes and four distinct camera angles on a Tesla Model S. For object visualization purposes, we chose to focus on the front camera view. During processing, we sampled one frame out of every eight and performed inference on those selected frames. For each processed frame within a scene, we extracted and stored relevant information—such as object type, position, and rotation—in a structured JSON format designed to support later rendering. This JSON file is then imported into Blender through its scripting interface. Using the details from the JSON, we place the corresponding 3D models—both provided and custom—into the scene, representing elements like vehicles, pedestrians, traffic lights, speed limit signs, and more.

II. Depth Estimation

Most important aspect of this project is to get good estimate of depth. There are many models which can give good and close to accurate results. However, we relied on the projection equation of a pinhole camera model to enforce relationship between expected height of objects and detected height of objects obtained using initial depth estimate from a depth map (obtained from ZoeDepth). We followed the following procedure:

1) We use ZoeDepth to get an initial estimate of metric depth.
2) We use camera pinhole equation and known constraint i.e height of most of sedan cars is in range 1.3m to 1.5 m.
3) Given the relationship between the height of the bounding box and the actual depth:
4) Based on this:

$$\frac{h_{\text{bbox}}}{f_y} = \frac{H_{\text{real}}}{Z} \tag{1}$$

Where:

- h_{bbox} is the height of the bounding box in pixels,
- f_y is the focal length in the y-direction (pixels),
- H_{real} is the real-world height of the car (meters),
- Z is the actual depth or distance to the object (meters).

5) Rearranging the equation to solve for Z:

$$Z = \frac{H_{\text{real}} \cdot f_y}{h_{\text{bbox}}} \tag{2}$$

6) Do this on every bounding box of car and get estimate of their depths and then calculate depth scale = actual depth/depth given by model
7) **Depth Ransac**- Project back the bounding boxes of cars into world and remove the outliers(occluded cars and bad boundary box) if the projected height is not close to the actual height range. Recalculate to get better depth scale and apply on all objects

Fig. 1: Sample depth map from ZoeDepth

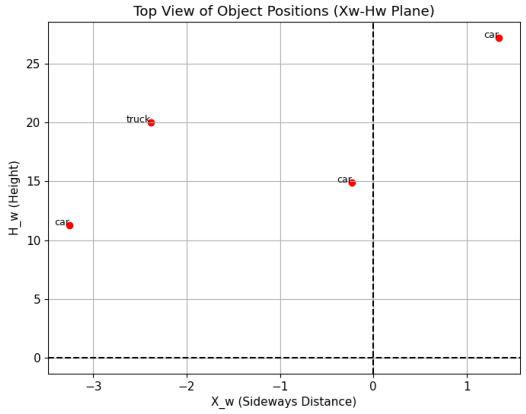

Fig. 2: Top-down view of relational depth between detected objects in a scene

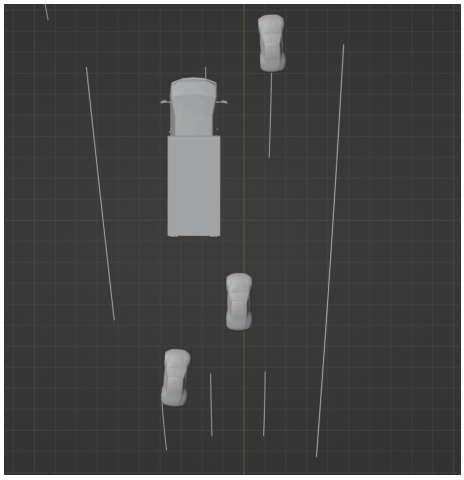

Fig. 3: Top-down view depicting relational depth between objects as rendered in blender

III. Lane Detection

We detect lanes using the following procedure:

- We take inference from MaskRCNN for the lanes
- Convert the image into binary image keeping only lane pixels as white and others black
- Then we fit best curve to get pixels co-ordinates for lanes.
- We assumed camera is mounted at height of around 0.9 m from ground and we used this constraint to get the depth of these points in world. We got exact matching in rendered frame.

Fig. 4: Road Signs with Lane Segmentation and Mask-RCNN

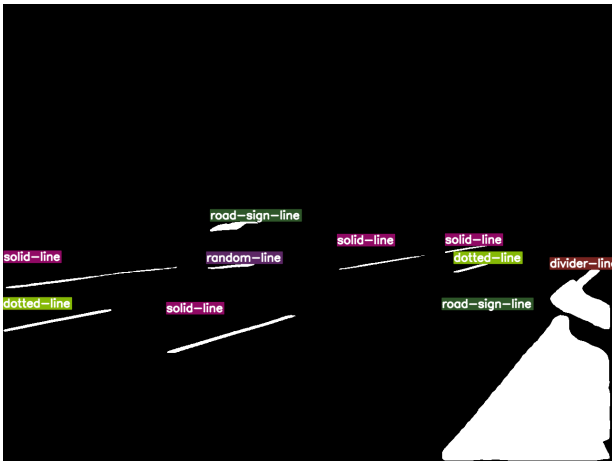

Fig. 5: Lane Segmentation using Mask-RCNN

IV. Object detection and Pose estimation

A. Object Detection

We initially used Yolo-world8 for vehicle detection but later found many classes are not defined in this model so we added detic model over it to get classes not covered earlier. With Detic, we were able to get various classes like pick-up truck, van, convertible, SUV etc.

B. Vehicle Pose estimation

We used the YOLO3D model to obtain object poses. However, we found that approximately 30 percent of the predicted poses were incorrect. To address this issue,

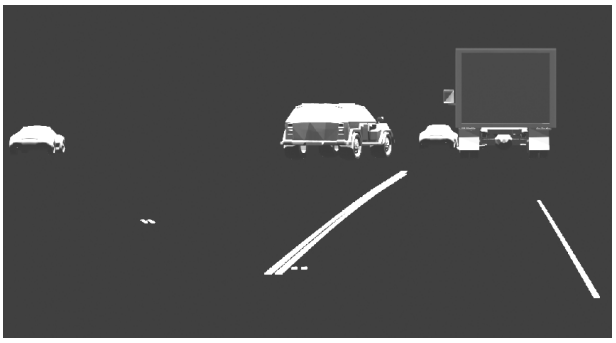

Fig. 6: Detection of cars, SUVs and trucks

we integrated PIFPAF, which provides key features of the objects. We performed feature binning using eight orientation bins based on these key features. All detected poses were then passed through the PIFPAF model to verify whether each pose aligned with the corresponding features. If the features did not align with the given pose, we recalculated the pose using the bin whose orientation was closest to the extracted features.

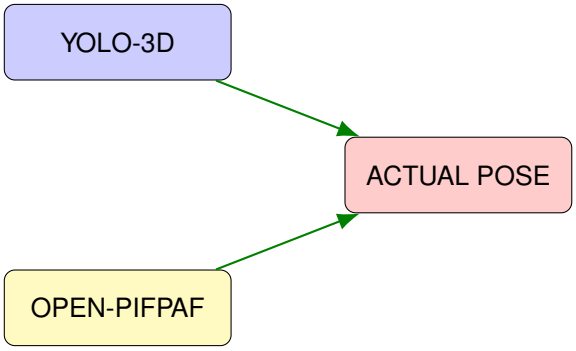

Fig. 7: Fusion of YOLO-3D and OpenPifPaf for pose estimation for calculating actual pose

We initially applied PIFPAF but found OSX model was directly giving better pose . OSX model output shown in figure.

C. Human Pose estimation

We initially used Pifpaf model but later found OSX provides better pose with direct inference.

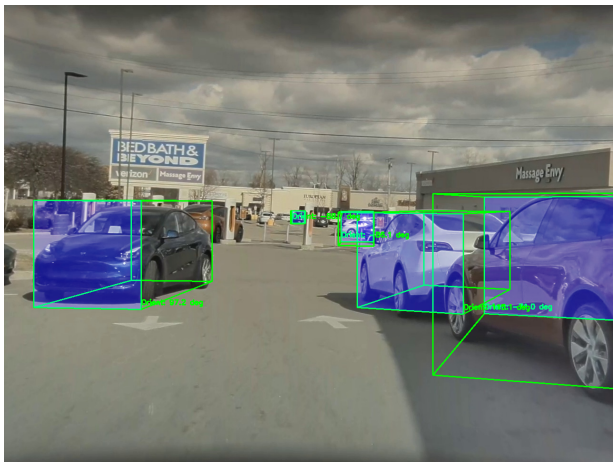

Fig. 8: Pose estimation using YOLO3D

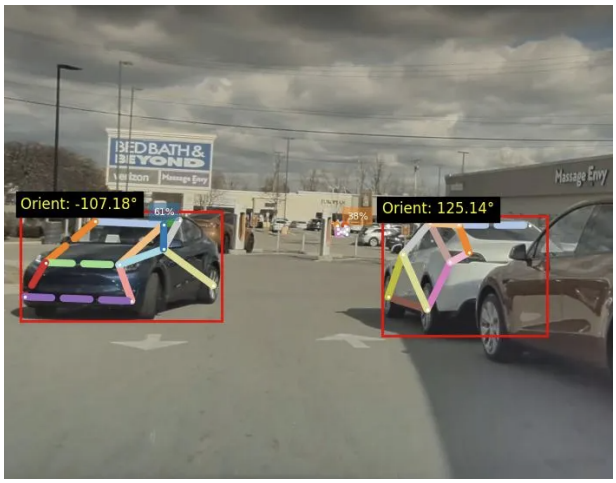

Fig. 9: Pifpaf Correction

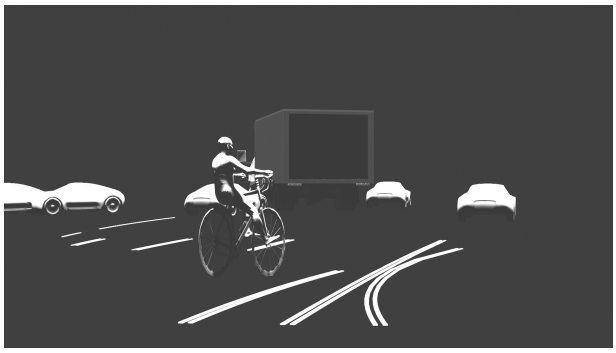

Fig. 10: Human Pose

V. Traffic Lights

Yolo-world8 gives the bounding box for the traffic light. We had these objectives - color detection shape of sign and direction of arrow.

A. Color-detection

We followed the following procedure for color detection:

1) Check aspect ratio of the box.If aspect ratio gives rectangle it means it is light with three lights and if it is close to square it is single light.
2) For traffic light with three lights, we divided the light into three parts .
3) For each part, we checked the brightness of patch at center.
4) Based on the bright area we decide the color (Red-Top, Middle-Yellow, Bottom-Green)
5) For single box we just performed hsv and decided color.

B. Shape of sign and direction of arrow

Shape detection procedure: Earlier we were using classical approach in which once we got the illuminate region, we were cropping the central patch of 60 percent area and then checking binary image patch in vertical and horizontal direction like shape of plus +, if we were getting white pixels on both vertical and horizontal, we assumed as circle otherwise arrow. Further, if we got arrow, we already knew whether horizontal or vertical from previous technique. Then we check centroid of white pixels to get exact direction of arrow . This technique looked promising but had many issues like the quality of bounding box given by model and whether light is at some angle which caused the center to shift and more of steel part of light come into picture.

1) ***Direct Inference method****:* We fed the patch of light to OCR model which we used for street sign reading and we noted that for up arrow it was showing ”1” or ”T”,for bottom arrow it was showing ”L”, for left ”F” and for circle it gave output 0 or o . For right we assumed ”-” but we could not test it due to no right arrow in videos.
It gave correct result for almost every light with single box and for 3 light box, it had good results for lights directly in front of it but in lights with larger lateral displacement the success rate was 50-60 percent . But the overall detection speed improved with less computation.

Fig. 11: Arrow detection using classical approach

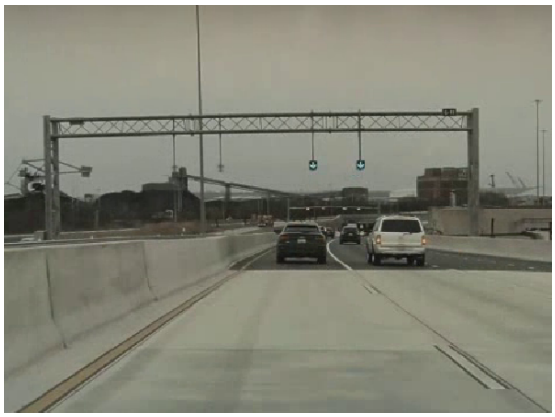

Fig. 12: (Top) Arrow detection using OCR. (Bottom) Actual Scene Frame

VI. Roadsigns on ground

- MaskRCNN detects the roadsign on ground
- Apply cv2.contour function on the detected roadsign mask to get pixel coordinates for the edges

- Project it into world using pinhole camera equation We were able to get almost exact shape of the roadsigns.

Fig. 13: Roadsigns on ground

VII. Streetlight Signs and other random objects

We used combination of yolo-world8 and detic to detect street signs particularly stop sign and speed limit sign. To read the speed limit and its value we used Florence2 OCR model.We also detected other objects like telephone poles, street light, trash can etc using DETIC.

VIII. Extra Credit

A. Brakelight and indicators

DETIC model detects blinker, tailight, brakelight. On the box given by detic, we check brightness using classical approach and if it is above a set theshold, we detect it as ON.We got accuracy over 90 percent in this. Only issue we faced was some wrong detection in case of glare.

B. Speed Bump Detection

On the detected street sign we applied same OCR model to detect the text "hump" and in that case we were able to find an approaching speed bump. It is pretty straightforward to put the speed bump beside the street sign on road but that was not the case as found in video 5. As noticed that all speed humps have some kind of lane marking, so we again used MaskRCNN to get check that marking. This procedure is summarized as follows:

- MaskRCNN detects the sign for speed hump as roadsign
- As discussed earlier, we already had binary image for that.
- As for a moving car, the roadsign mark for speed hump is always close to horizontal center in the image we used this condition for detection.
- So get a small sliding window of 20 pixels at width i.e $center_x$-20 to $center_x$+20 .
- Then we slide this single pixel depth bar from bottom to top to get y i.e row coordinates of the detected sign in image
- For the first white pixel detected, we note it as row coordinates of the hump in image
- Then we get world location of that point in world using pinhole equation as we assumed Y in world as 0.9 m .
  We were able to successfully detect the speed bumps in all cases.

C. Detection of Motion

We used RAFT algorithm for detection of motion. But using it directly applying threshold did not give us helpful results as it gives relative motion.Another available method was calculating sampson threshold which was very expensive to do on each image.

1) Our Method: Get idea of speed of the self using static items.We used detic model to get static objects like street signs, poles, dustbin etc. As image pixel flow output given by RAFT model also depends upon depth we decided to move everything to world coordinates. Procedure is shown below and mathematical reasoning is provided in later section alongwith and collision detection.

- Get relative motion of self in real world which gives estimate of actual velocity of self in real world (opposite direction)
- Used RAFT model to detect moving cars in same direction(lower raft output), moving cars in opposite direction (higher raft ouput in downward direction), parked cars (close to static objects)

D. Collision Detection

Below we explain the approach which we took for this detection . Output was not good as expected as dependency of equation on exact depth and RAFT output both of which are not very accurate.

E. Relative Depth Estimation and Collision Prediction

We start with the following equations:

We begin with the standard pinhole camera model equation:

$$\frac{h_{\text{bbox}}}{f_y} = \frac{H_{\text{real}}}{Z} \tag{3}$$

From this equation, we can rearrange to find the depth Z (which we denote as $z_w(h_{\text{bbox}})$):

$$z_w(h_{\text{bbox}}) = \frac{H_{\text{real}} \cdot f_y}{h_{\text{bbox}}} \tag{4}$$

Since $H_{\text{real}} \cdot f_y$ is a constant (for a given object class), we define:

$$z_w(h_{\text{bbox}}) = k_p \tag{5}$$

Now, considering motion, as pixels move to a new position, the depth changes. The new depth z_n (from the new bounding box) is approximately equal to the old depth z_w:

$$z_n \approx \frac{z_w(\text{old bbox})}{\text{new bbox height}} \tag{6}$$

Let the old bounding box height be approximated using depth flow values as:

$$\text{old bbox height} = |v_2 - v_{\text{dp}} - (v_1 - v_{\text{up}})| \tag{7}$$

where: - v_2 and v_1 are pixel positions of the bottom and top of the bounding box, - v_{dp} is the depth flow at the bottom of the bounding box, - v_{up} is the depth flow at the top.

We take the absolute value because the negative sign only indicates direction, and we are only interested in magnitude (we already conditionally consider only the downward flow):

$$\text{bbox}_{\text{effective}} = |v_{\text{db}} - v_{\text{up}}| \tag{8}$$

Assuming the optical flow output represents the rate of pixel movement, we treat the new depth after a unit time as:

$$z_n = z_w \left(1 - \frac{v_{\text{du}} - v_{\text{db}}}{v_2 - v_1}\right) \tag{9}$$

So, the net velocity is the change in depth per unit time:

$$v = |z_n - z_w| = z_w \cdot \left|\frac{v_{\text{du}} - v_{\text{db}}}{v_2 - v_1}\right| \tag{10}$$

To maintain safety, we assume a reaction time of 5 seconds (including a safety factor). Thus, the **dangerous condition for collision** becomes:

$$z_w \leq 5 \cdot v \tag{11}$$

i.e.,

$$z_w \leq 5 \cdot z_w \cdot \left|\frac{v_{\text{du}} - v_{\text{db}}}{v_2 - v_1}\right| \tag{12}$$

which simplifies to:

$$1 \leq 5 \cdot \left|\frac{v_{\text{du}} - v_{\text{db}}}{v_2 - v_1}\right| \tag{13}$$

Hence, the **collision danger condition** is:

$$\left|\frac{v_{\text{du}} - v_{\text{db}}}{v_2 - v_1}\right| \geq \frac{1}{5} \tag{14}$$

This approach allows us to use depth flow estimates for real-time collision prediction based on the observed motion of objects in the scene. Note that we make the following assumptions:

- We get perfect bounding box so we take patch at top and bottom instead of single pixel
- We have very good estimate of depth
- RAFT output is rate in SI units for the movement of pixel

F. Conclusion

Entirely working in world coordinates helped us to get better results and pixel motions have velocity and depth entangled together. We tried on two videos and we were getting rational results and for few scenes it could also sense higher risk . But we could not find the condition where we had time less than 5 sec to react so just to showcase our method we increased the respond time to 15 secs which gave us few results.

IX. Visual Results

Fig. 14: (Top) Original scene for capturing optical flow. (Bottom) RAFT optical flow output.

Fig. 15: Parked Vehicle Detection

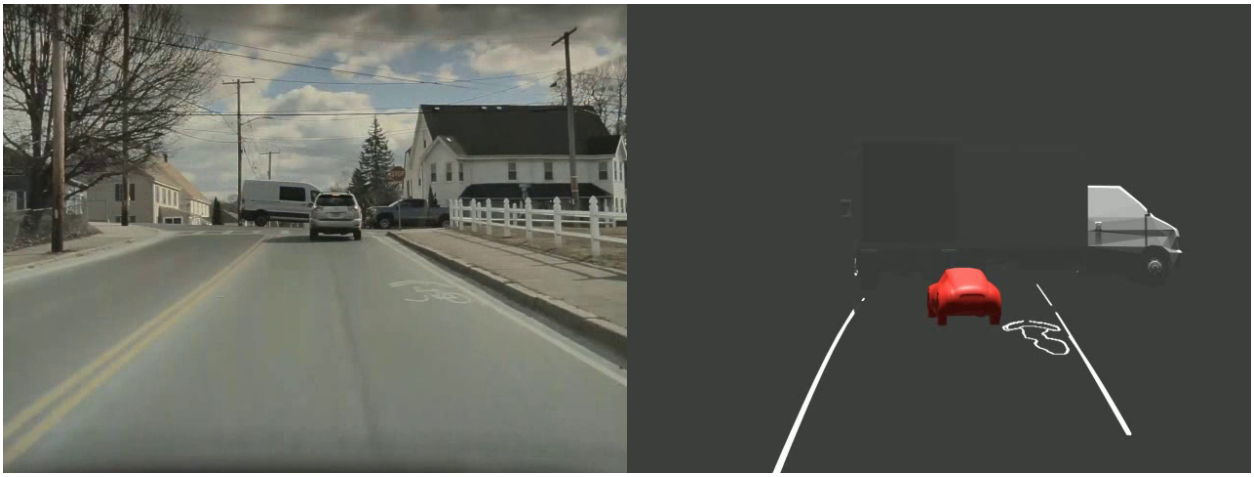

Fig. 16: Detection of imminent collision

Fig. 17: Stop sign and road marker detetion

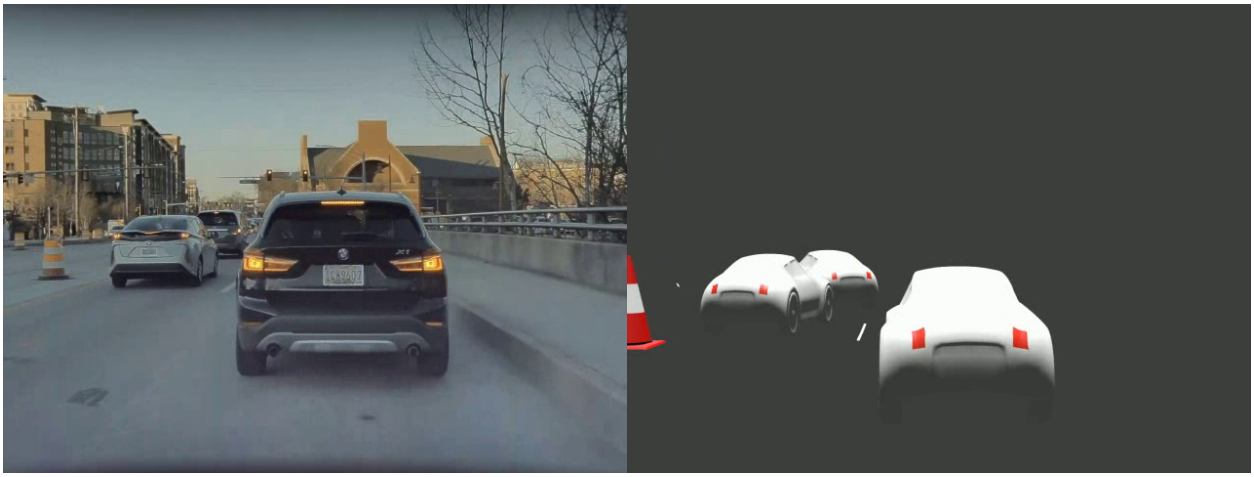

Fig. 18: Break Light Detection

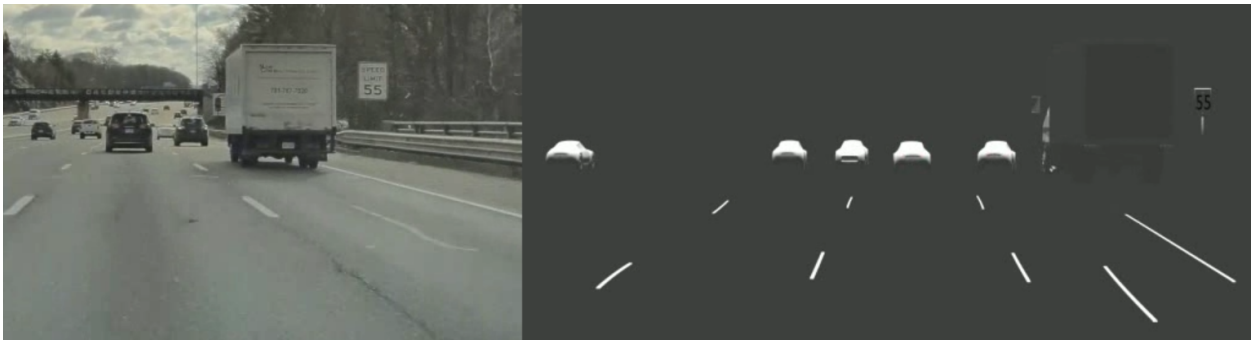

Fig. 19: Speed limit sign detection

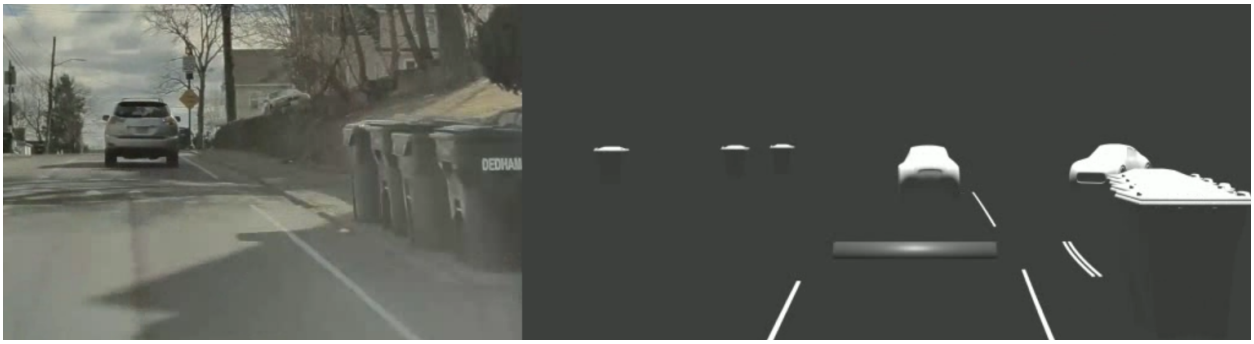

Fig. 20: Speed bump detection

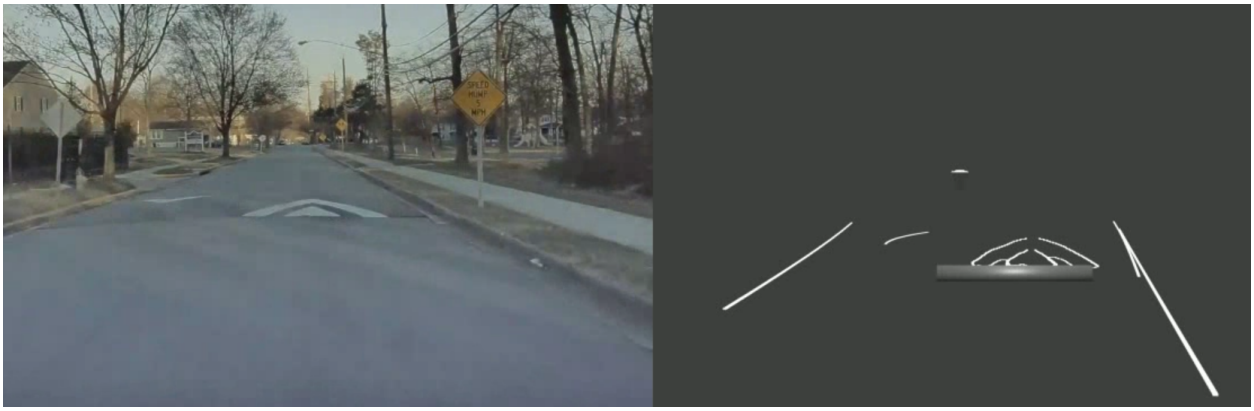

Fig. 21: Speed bump detection