Project 3: Tesla Vision

Dhiraj Kumar Rouniyar
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, Massachusetts

Dhrumil Kotadia
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, Massachusetts

Using four Late Days for Project 3

INTRODUCTION

This project aims to enhance and reimagine Tesla's dashboard visualization for the 2023 Model S. Inspired by Tesla's innovation, we have attempted to create a visual dashboard display using Blender. Our goal is to produce a rendered video that showcases the front-facing camera view alongside dynamic representations of the vehicle and its surroundings. The project is divided into three phases. We shall discuss the work in each phase in detail in the following sections.

BACKGROUND AND DATASET STRUCTURE

For the dataset, we have the 13 videos representing 13 scenes captured from the 4 cameras of the Tesla models S. The original distorted and the undistorted videos both have been provided along with a calibration sequence for the camera. The dataset also contains a markdown file containing all the objects present in the scenes. We have divided these videos into frames and worked with them to render a blender video. Calibration data was available and K matrix was calculated from the calibration data.

PHASE1

In phase 1, the following basic features were implemented for the project:

- Lanes: Different kinds of lanes on the road: solid, dashed, double lines and colour of the lanes were detected and implemented. This was actually implemented in phase 2 completely. In phase1 we had implemented the YOLOPV2[1] pretrained model to detect the lane lines and the driveable area. The shortcoming here was that the pretrained model only provided the masks for the lanes and the driveable area. The model did not provide the actual differentiation between different kinds of lanes. Thus, in phase2, Mask-RCNN[2] was implemented. This model provides a bounding box for each lane and an overall mask showing the lanes. It differentiates between the types of lanes as well. The shortcoming of this model is that it does not identify the color of the lane. For this, we resorted to classical methods using OpenCV to detect the colors.
  For each bounding box of the lane marker, we cropped the image to the size of the bounding box to ensure that we would only have the lane marker in the image. We also did the same to the resulting mask provided by the model to obtain the mask of the cropped image. We will call this mask1. We dilate mask1 to obtain mask2. Then we subtract mask1 from mask2. The core idea here is to generate a mask of the road surrounding the lane marker. Once we have the Hue, Saturation and Value of the road, we can compare it with that of the lane marker. If the lane marker has more saturation than the road, it indicates that the lane marker is yellow. The lane marker is considered white in all other cases. The process can be seen in the image 1

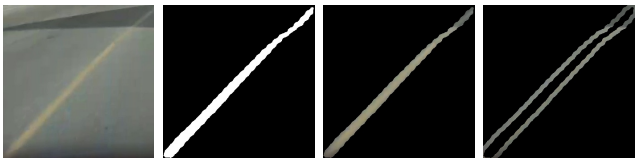

Fig. 1. Lane Color Detection(Left to Right): Lane image, Lane mask, Masked Lane Image, Masked Image of the road surrounding the Lane

The resulting lane in the simulation can be seen in image 3

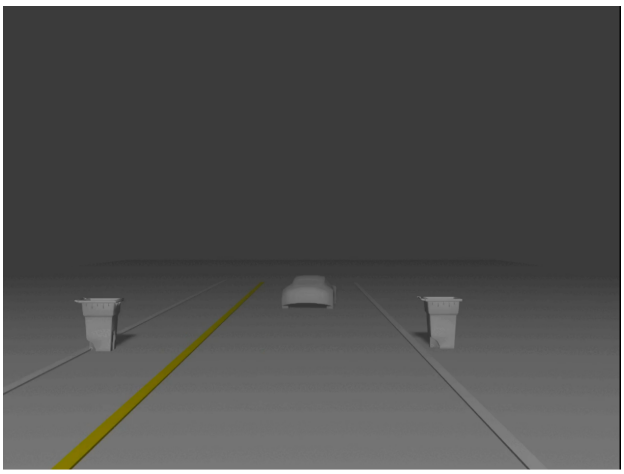

Fig. 2. Actual Image and Blender Simulation

- Vehicles: Here, we identify all the cars using YOLOv8 model trained with COCO dataset and represent them as car shapes in the simulation. This is visible in image 3

Fig. 3. Object Detection using Detic

- Pedestrians: Pedestrians were identified in the actual images using YOLOv8[3] and rendered them in blender but did not idOentify the pose of the pedestrians. It should be noted that we replaced YOLOv8 with Detic[4] in later phases. Default poses were used for all pedestrians for this phase.
- Traffic Lights: In phase 1, traffic lights were identified using YOLOv8. The colors of the traffic lights were identified using classical methods. YOLOv8 would provide the bounding box for the traffic light. The bounding box was used to crop the image and apply two separate filters generating separate images to extract the amount of red and green colors each in the traffic lights. Once they are obtained, both of them were converted to grayscale and contours were obtained. The resulting contours were eroded and dilated once to get rid of unnecessary noise. After this, out of the red and green contours, the contour with the largest area decides the actual color of the traffic light. This can be visualized in image 4

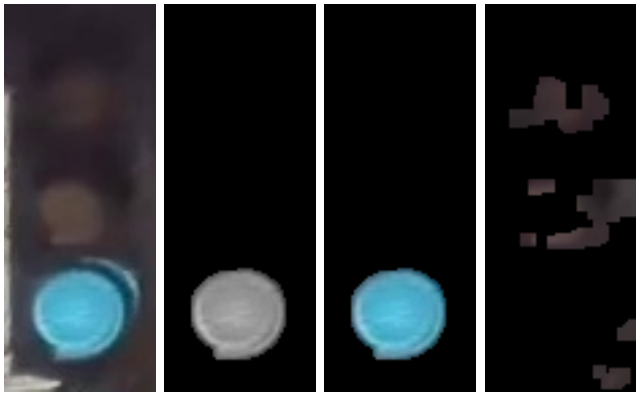

Fig. 4. Traffic Sign Color Detection(Left to Right): Cropped image, Grayscale Image, Green Masked Image and Red Masked Image

- Stop Sign: In this phase, the stop sign is identified using the YOLOv8 model. The resulting image can be seen in figure 5

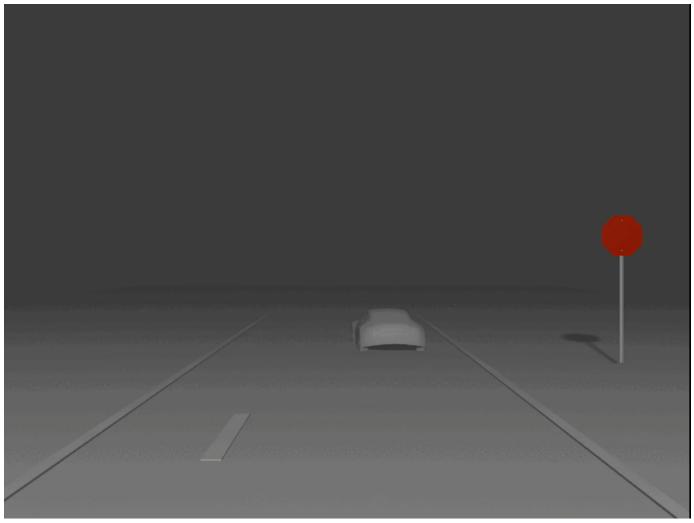

Fig. 5. Image showing the render of stop sign in phase 1

PHASE2

In phase 2, the following advanced features were implemented for the project:

- Vehicles: In this phase, we identify the different types of vehicles i.e. Cars, Trucks, Bicycle using YOLOv8 model trained with COCO dataset. Since the COCO dataset does not provide classification between sedan, SUV and pickup trucks, in phase3, the model was changed to a pretrained Detic model trained with Objects365 dataset. This model provided the necessary classification between all vehicles. This is evident in figure 22.
- Traffic Lights: In phase 2, on top of the detection of traffic lights and colors using YOLOv8 arrows were detected and rendered. The color of the traffic lights was detected as done in phase 1. Once the color is identified, if the color is green, the contours obtained for the green light were eroded and dilated to obtain a noise free contour. After this, the contour obtained is the matched to the previously saved contours of the arrow directions using cv2.matchshapes. For output, this function provides the distances between each pair of shapes. The pair with the least distances indicates the arrow direction. If all distances are more than a given threshold (0.25 - decided after various runs), it is decided that the traffic light does not have an arrow. This can be visualized in image 6.

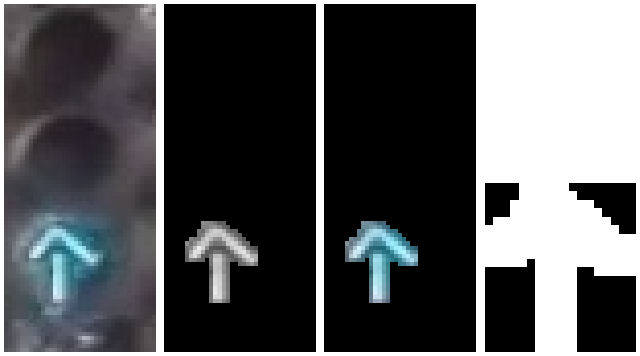

Fig. 6. Traffic Sign Arrow Detection(Left to Right): Cropped image, Grayscale Image, Green Masked Image and the Resulting Contour

- Road Signs: In this phase, the road signs (Arrows) are identified. Here, we make an assumption that only 1 road sign is visible. For detection, we obtain the driveable area mask from the YOLOPv2 model. Using the method described for lane color detection, we identify the road color. We generate a mask to isolate the colors in the driveable area. For that, we convert the images to HSV and keep only the colors that have a high 'V' value in the driveable area image. Once we have obtained this, we convert the image to grayscale and obtain the contours. Once we have the contours, we select the contour with the largest area and match it with the pre-saved contours. As described in the traffic light arrow detection, we match the contours and obtain the direction of the arrrow. The entire process can be seen in figure 7 and 8

Fig. 7. (Left to Right)Original Image from Scene 7, Isolated signs on the road and the Resulting largest Contour obtained

Fig. 8. (Left to Right) Pre-Saved contours Up, Left, Right and Down

In phase 3, we changed the pretrained model from YOLOPv2 to Mask-RCNN. Mask-RCNN provides bounding boxes for each lane as well as road sign lines. Based on the bounding boxes obtained we crop the image and identify the arrow sign using the method mentioned above. The only difference is that instead of applying it on driveable area, we apply it to a bounding box in Mask-RCNN. This method also has many false positives where many road crossing lines are detected as up arrows.

- Objects: In phase2, separate networks were used for detection of objects like traffic cone, drum and traffic pole - [5], trash bins - [6], speed Limit signs - [7]. The fire hydrant is detected using the YOLOv8 model. In phase3, we have updated from YOLOv8 to Detic trained with Objects365 dataset. Because of this, we have removed all separate models as Detic itself now identifies all objects.
- Pedestrians: In phase 2, the pedestrian poses are identified using the model. The resulting poses rendered can be seen in figure 9.

Fig. 9. Rendered Images showing pedestrian Pose and Trash Bin

PHASE3

In phase 3, we focus on some other details like brake lights and indicators of the cars as well as showing parked and moving vehicles using optical flow. This is discussed as follows:

- Brake Lights and Indicators: For detection of brake lights, we use the bounding boxes and the masks for vehicles generated by Detic. We crop the image using the bounding box and remove the parts of the image that are not present in the car mask provided by Detic. After obtaining this, we convert the image to HSV. After that, we apply a saturation and brightness filter to isolate the tail lights and identify if the lights are on. This is evident in figures 20, 13, 14 and 15.

Fig. 10. (Left to Right) Original Image from Scene 11, Isolated masked car Image and the filtered image to identify brake lights

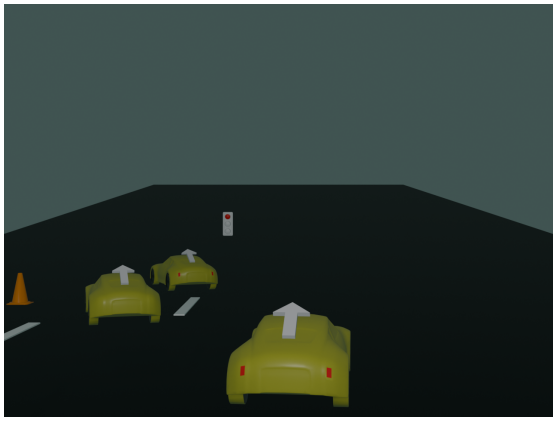

Fig. 11. Rendered tail light and red traffic light image example

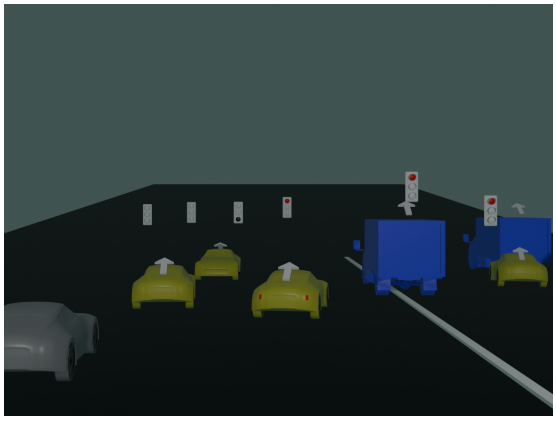

Fig. 12. Rendered tail light and multi traffic light image

Fig. 13. (Left to Right) Original Image from Scene 11, Isolated masked car Image and the filtered image to identify brake lights

Fig. 14. (Left to Right) Original Image from Scene 11, Isolated masked car Image and the filtered image to identify brake lights

Fig. 15. (Left to Right) Original Image from Scene 3, Isolated masked car Image and the filtered image to identify brake lights

Identifying the turning indicators was attempted using the classical methods but it was difficult to differentiate between brake lights and turn indicators due to different ambient lighting conditions. This is a limitation which could be solved using a machine learning model.

- Traffic Cones, Traffic lights, Lanes and Car Direction:

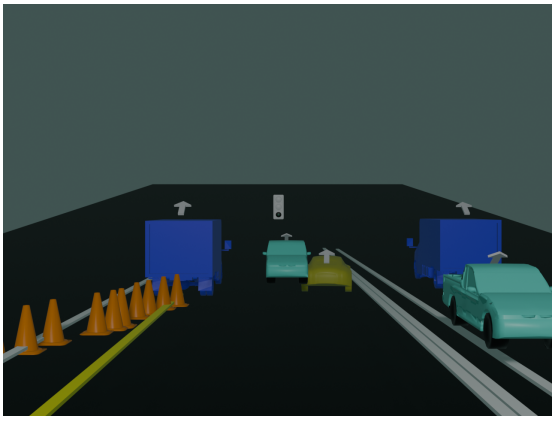

Fig. 16. Rendered image 1

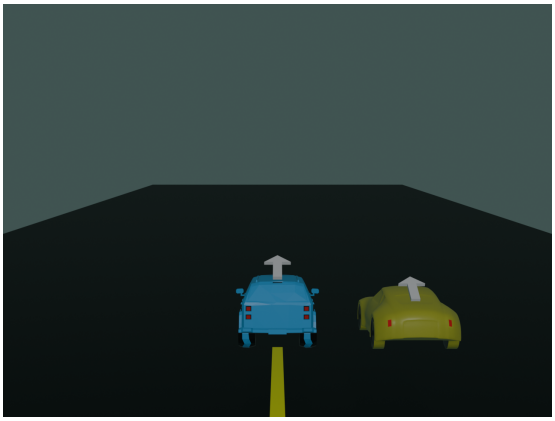

Fig. 17. Rendered image 2 - SUV and Car

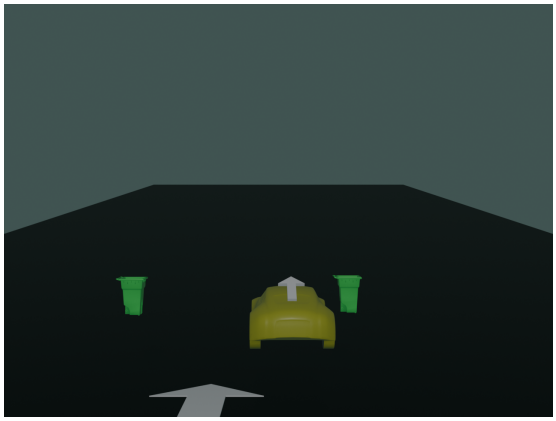

Fig. 18. Rendered image 3 - Road marker

- Speed Limit and Zebra lanes:

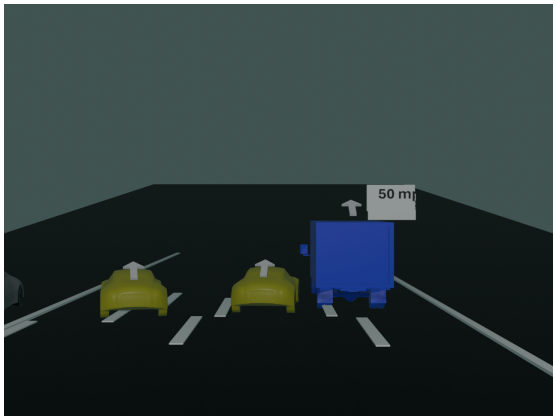

Fig. 19. Rendered image

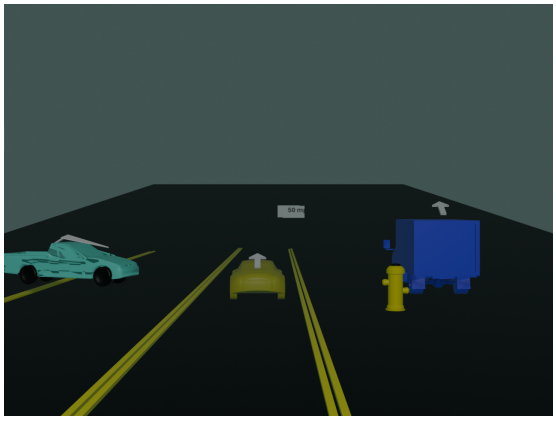

Fig. 20. Rendered image - Hydrant, Speed Limit, Break lanes

- Parked and Moving Vehicles: Parked and moving vehicles were identified in this phase using optical flow. Optical flow is calculated using OpenCV (cv2.calcOpticalFlowFarneback) for the frames and the resulting displacement is recorded. A fixed number of points(20) are selected in the scene on far left and far right of the image. If the displacement of these points is not zero, it is considered and the mean of the norm of all these displacements is taken. This provides a threshold for the frame. Next, we consider the bounding boxes of the vehicles and find the displacement of the particular vehicle. Once this displacement is obtained, it is compared to the previously calculated threshold. If the absolute difference between these two values is less than a pre-decided value (2 in our case), the vehicle is considered to be in motion. Otherwise, the vehicle is considered to be parked. Also, direction of the vehicles is also displayed with arrows on top of the vehicles. This can be visualized in figures 21 and 22.

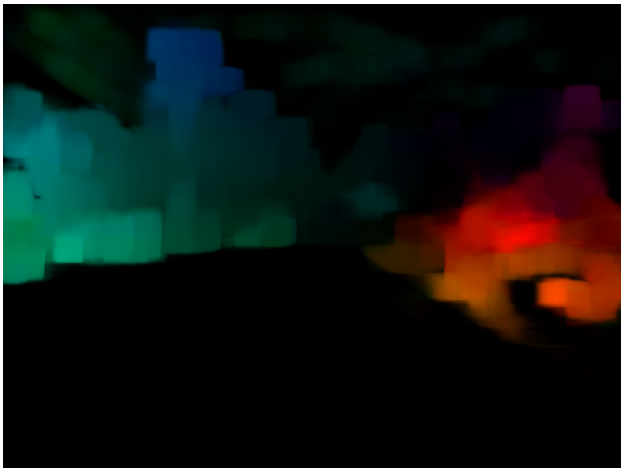

Fig. 21. Optical Flow for a Frame in Scene 5

Fig. 22. Rendered frames showing parked(Gray) and Moving(Yellow) Vehicles and Direction of motion of the vehicles(Arrow on top)

Extra Credit

For extra credit, we have implemented detection of speed humps and vehicles that have possible collisions. This is discussed as follows:

- Speed Humps: For detection of speed humps we have used the model yolov7 - [8] modified. This can be observed in figure 23.

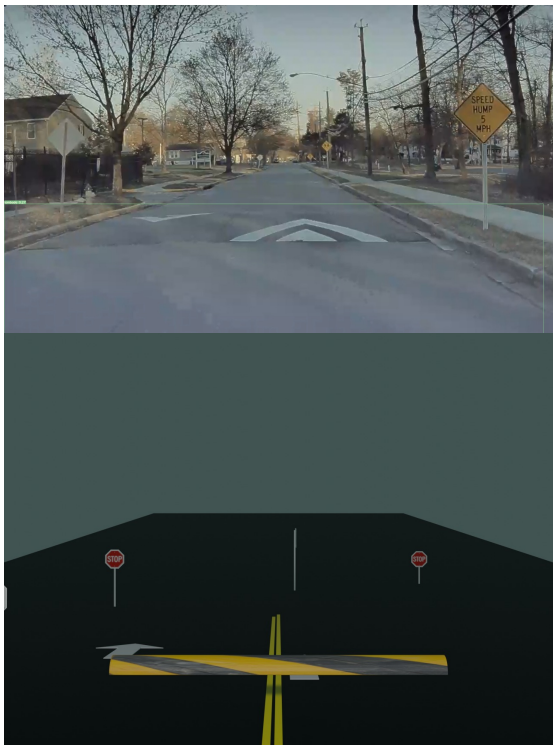

Fig. 23. Detected Bounding Box and rendered Speed Hump

- Collision Detection: Optical flow calculation done in phase 3 has been used to predict collision detection. From the optical flow, we obtain the future prediction of vehicles in the frame. Using depth, we obtain the Z coordinate of the future predicted position. With this, we calculate the distance between camera and the car in 3D and if the distance is less than a pre-decided threshold, we highlight the car to show a predicted collision.

CONCLUSION:

It can be concluded that while we were able to generate a visualization in blender by using the various deep learning models and classical methods, the visualization itself is far from desirable. The inference time for deep learning models is very high and all classical methods implemented have a lot of false positives and negatives which make it unusable and it cannot be implemented on real vehicles. Specially trained models can highly improve the output and thus can make the system usable but in it's current state, it cannot be used.

REFERENCES

[1] https://github.com/CAIC-AD/YOLOPv2.
[2] https://debuggercafe.com/lane-detection-using-mask-rcnn/.
[3] https://github.com/ultralytics/ultralytics.
[4] https://github.com/facebookresearch/Detic.
[5] https://universe.roboflow.com/flood-lzlds/trafik-konileri/model/1.
[6] https://universe.roboflow.com/cyfbxl/-vqmac/model/1.
[7] https://universe.roboflow.com/us-road-signs-projects/us-road-signs.
[8] https://colab.research.google.com/github/jpscard/Deteccao-de-buracos-e-lombadas-com-yolov7.