RBE 549 Project 3: Einstein Vision

Aabha Tamhankar
Masters in Robotics Engineering
Worcester Polytechnic Institute
astamhankar@wpi.edu

Miheer Diwan
Masters in Robotics Engineering
Worcester Polytechnic Institute
msdiwan@wpi.edu

Using 5 late days

***Abstract*—In this project we created visualization for an autonomous vehicle inspired by Tesla's dashboard. We combined different deep-learning and classical computer vision techniques and seamlessly integrated them into our pipeline to address each of the problems posed to autonomous driving vehicles.**

I. Introduction

The development of self-driving cars has been one of the most significant technological advancements of recent times. As these vehicles become more prevalent, it is becoming increasingly important to create an intuitive, easy-to-use dashboard for drivers and passengers. The dashboard of a self-driving car needs to display information about the car's surroundings, as well as its own status, in a clear and easily understandable way. The project is based on Tesla's dashboard HMI (Human-Machine Interface) is designed to provide a seamless and intuitive experience for the driver, enabling them to interact with various features and functions of the car while driving. Given video sequences from a Tesla car, the aim of the project was to render visualizations of these sequences through Blender.

II. Dataset

The data contains the following:

1) 13 video sequences under various environment conditions. These videos are presented in raw as well as undistorded using calibration copies.
2) Calibration videos which can be used to calibrate the cameras.
3) Blender Assets for various things like Cars: Sedan, SUV, Pickup truck, Bicycle, motorcycle, Truck, Traffic signal, Stop Sign, Traffic Cone, Traffic Pole, Speed Sign and Pedestrian. Texture images for stop sign and a blank speed sign are also included

III. Perception and Detection of the Environment

Object detection is an essential component of any self-driving car dashboard. The ability to accurately detect and classify objects in the car's surroundings is critical for ensuring safe operation of the vehicle. Object detection involves using identifying and locating objects in the car's environment, such as other vehicles(cars, trucks, SUVs) ,pedestrians, and traffic signals, stop signs, etc. It is neccessary to find the pose of all these objects in the environment.

A. Object Detection and Classification

YOLO (You Only Look Once) is a popular object detection model known for its speed and accuracy. It was first introduced by Joseph Redmon et al. in 2016 and has since undergone several iterations, the latest being YOLO v8. YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. You Only Look Once (YOLO) proposes using an end-to-end neural network that makes predictions of bounding boxes and class probabilities all at once.

For this project, we are using YOLO v5 which is used for detecting and classifying objects as well as perform semantic segmentation on the detected objects. YOLO v5 can detect up to 80 classes including cars, trucks, bicycles, traffic signals, stop signs and pedestrians. It also generates 2D bounding boxes around the detected objects. We extracted the coordinates of the bounding boxes using this model and stored them in a '.csv' file. We did not use the semantic segmentation data generated by the model. Semantic segmentation could have been used to resolve an issue with multiple vehicles spawning at the same location in Blender. However, we resolved this issue with a different trick as shown in IV-B. These coordinates were later used to estimate the relative depth of the detected objects as seen in III-B and to spawn objects in Blender as seen in IV.

The resulting images of running the model for our videos are illustrated in Fig.1

B. Monocular Depth Estimation

Humans perceive depth in a scene by using visual cues such as perspective, occlusion, and shading. Similarly, depth maps generated by the MIDAS model can be used to provide depth information to a computer vision system. MIDAS (an acronym for "Mi"llion "D"epth "A"nnotations "S"amples) is a deep learning model designed for depth estimation, which refers to the process of predicting the distance of each pixel in an image from the camera that captured it.

The MIDAS model is based on a deep neural network architecture called ResNeXt, which has been pre-trained on a large dataset of natural images to extract high-level features from input images. The MIDAS model uses these pre-trained features as inputs and trains additional layers to perform depth estimation. The training data for MIDAS consists of millions of RGB-D (red-green-blue and depth) image pairs. These image pairs are used to train the model to learn the

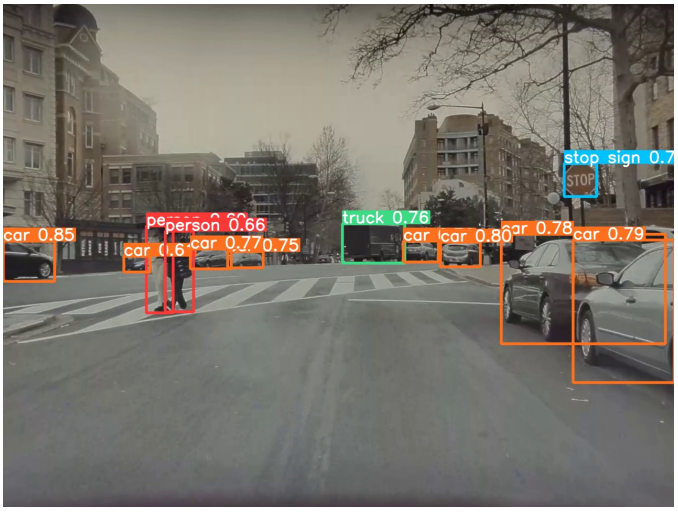

Fig. 1: Object Detection Model Output

relationship between the visual features in an RGB image and the corresponding depth values.

The output of the MIDAS model is a depth map, which is a 2D image where each pixel represents the estimated distance from the camera that captured the input image. The depth map is typically a gray scale image, where darker pixels represent objects that are closer to the camera, and lighter pixels represent objects that are further away. To find the relative depth of objects in the image, we first find the minimum and the maximum depth value of the image. We then find the average depth value of the crop of the object using the coordinates obtained from the YOLO v5 network. The scaled depth is obtained by taking the absolute value of the difference of the average value of the cropped region and the maximum depth value of the image.

By using the depth map, it is possible to extract spatial information about the objects present in the scene, which can

Fig. 2: Depth Map Obtained Using MiDaS

aid in understanding and analyzing the visual content.

C. Lane Detection

The first step in this project involved lane detection, lane segmentation and drivable region segmentation. Initially, we tried to implement pre-trained networks such as YoloP and HybridNets to detect lanes and segment the drivable region. However, the results we got were very inaccurate.
After trying and failing on numerous deep learning networks for detecting lanes, we decided to stick with classical method of lane detection. In our classical approach, we used the OpenCV function 'cv2.HoughlinesP' to detect lines in the image. This function returns the starting point (x1, y1) and the ending point (x2, y2) of the detected line.

1) Algorithm:

- Convert original image to grayscale.
- Apply gaussian blur on the grayscale image.
- Use canny edge detection on the input image to isolate the edges.
- Use the cv2.HoughlinesP function to detect lines in the image and overlay them on the original image.

2) Issues with Classical Approach:

- This approach is not robust and relied heavily on parameter optimization. The parameters of the canny function and the HoughlinesP function needed to be tuned for different frames to get accurate results.
- It also depends on the quality of the input image and the condition of the paint used for drawing lanes on the road in the real world. Even after heavy parameter tuning, the results we achieved were barely passable. However, it gave us better results than the deep learning approaches that we found.
- Horizontal lines were also being detected. We fixed this by adding a constraint on the slope of the detected line. If the slope was < 0.5, the line was ignored.

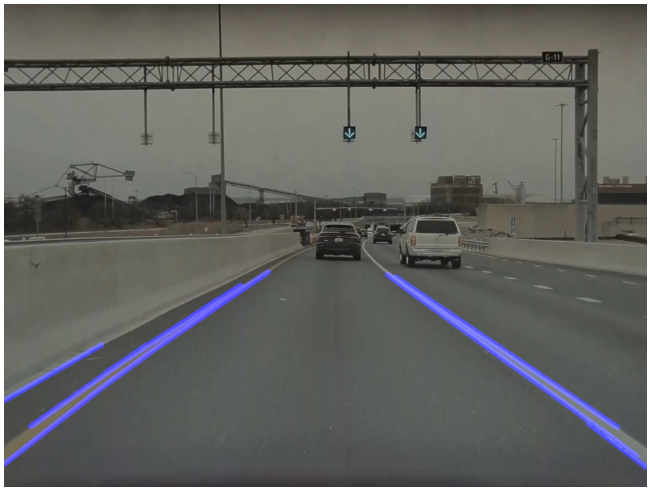

Fig. 3: Output of Lane Detection

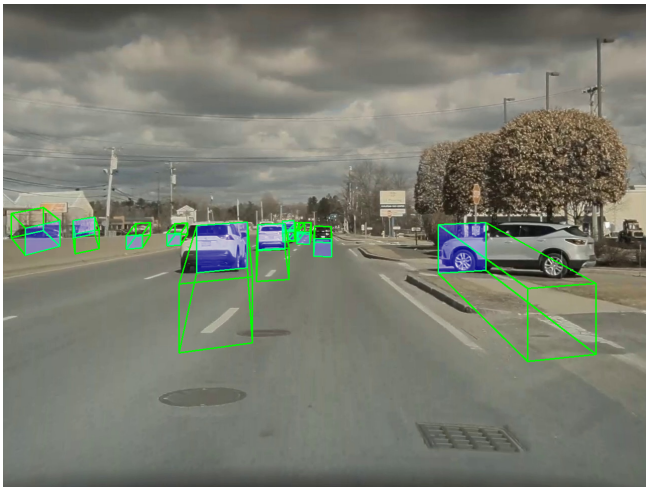

Fig. 4: Output of YOLO 3D

- Multiple lines are detected on a single lane and if you try to render all of these in Blender, it creates a mess. To avoid this, we only considered the minimum (left most) and maximum (rightmost) 'x' coordinates and spawned lanes at these locations in Blender.
- Lines were being detected in the upper half of the image where there were no lanes. We solved this issue by taking a binary mask and only considering the lower half of the image where we knew lanes existed. This idea was later in the project for road sign detection where we only considered the upper half of the image as this is where the sign were more likely to be found.

3) Future Improvements to the Lane Detection Approach:

- Using better Neural networks such as YoloPv2 could be used to get more accurate lanes.
- Combining classical and deep learning approaches could yield better results.
- Using color thresholding such as HSV and HSL in OpenCV to segment yellow and white lanes.

D. 3D Vehicle Pose Estimation

Vehicle pose estimation is crucial for any autonomous vehicle as it helps to identify the orientation of the objects in the environment. We tried to use a pre-trained network called YOLO 3D for this task. However, the results produced by the model were very inaccurate as seen in Fig.4 So, we decided to do the vehicle poses manually in Blender.

1) Special Case: This approach only works for horizontally oriented vehicles as seen in 'Sequence 5' provided in the dataset. Our idea was to use the pixel length of the vehicles in the image to determine the pose of the vehicle. Since a horizontally oriented vehicle would have a larger pixel-area in the image, we could spawn vehicles horizontally if the pixel length was above a certain threshold. However, only the pixel length was not enough since a vehicle would have smaller pixel areas at farther distances in an image. So, we decided to use the depth value of the vehicle multiplied by the pixel length of the object. If this value was above a certain threshold we spawned the car horizontally.

IV. Visualization

Blender is an essential tool for anyone involved in 3D creation and animation, whether it be for personal projects or professional work. With its vast array of features and tools, and its active community of users, Blender is a powerful and accessible tool for creating high-quality 3D content.

A. Rendering Objects

The results of YOLO v5 model were in the form of extreme x and y co-ordinates of the detected bounding boxes. Our rendered model spawns the center of the detected objects at the average of these minimum and maximum values. In blender, Y is the horizontal co-ordinate and Z is the vertical co-ordinate which are achieved through the YOLO v5 model. In order to avoid multiple cars spawning at the same location, we added an offset (= the length of the car asset) to the depth value.

B. Depth

Depth is perceived as a 3D point cloud. Since we already have xy co-ordinates, we can find the depth value of the object of interest. This depth value is then scaled according to rendered model and used as the X co-ordinate of the render.

C. Lanes

The extreme co-ordinates of the detected lanes are considered for the rendering of the lanes. Hence, the right and left most lanes are being rendered in the videos.

D. Human Armature

The human asset provided is in T-pose which is not the real pose of human beings. To make a more realistic human figure, armature function in blender was used. The result is provided in Fig. 5.

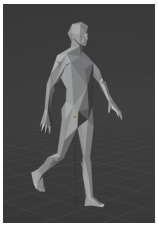

Fig. 5: Armature for Human Asset

V. Results

Fig. 6: Outputs for Scene 5

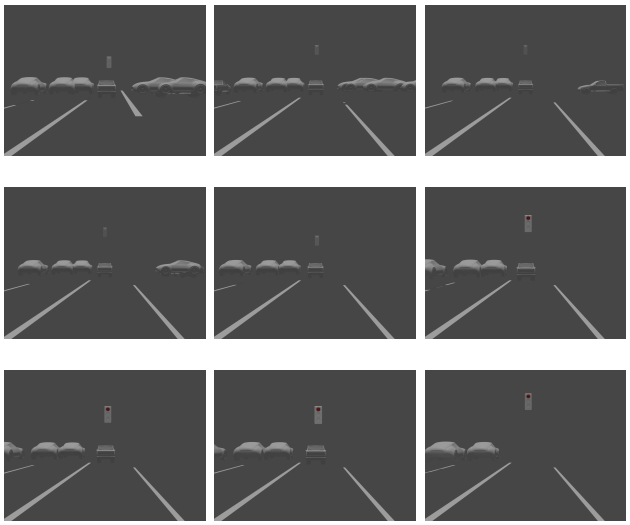

Fig. 7: Final Renders for Scene 4

References

[1] YOLO v5 Github Issues
[2] Roboflow Computer Vision Models
[3] Depth Estimation
[4] PyTorch MiDaS
[5] Udacity Nanodegree Project 1