Project 3 : Einstein Vision

Abhijeet Sanjay Rathi
M.S. Robotics
Worcester Polytechnic Institute
Email: asrathi@wpi.edu
Using 3 Late Days

Anuj Jagetia
M.S. Robotics
Worcester Polytechnic Institute
Email: ajagetia@wpi.edu
Using 3 Late Days

***Abstract*—This project entails the thorough implementation of creating a realistic road scenario visualization using camera sensor data from an autonomous vehicle. The pipeline from beginning to end consists of detecting road objects and using Blender 4.0 software to visualize them. We used a variety of object detection models, including Detic, OSX, Zoe Depth and Mask RCNN, that were pre-trained using various algorithms and network architectures. The introduction section provides details on the three distinct phases that comprise the entire project.**

I. Overview

Our method started with the camera being calibrated in order to extract the camera intrinsic and recover the unprocessed raw output. The undistorded footage was then subsampled to produce number of images. We took every 10 frames in the video, a key frame and gave that keyframes to the models that are used for the scene semantic rendering.

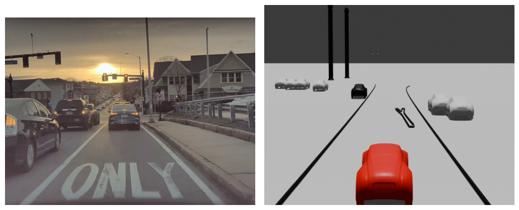
II. Blender Rendering

To render the modeled scene semantically, we used Blender as the rendering engine. We utilized Python scripting to reconstruct the scene for each frame in the JSON file, which contained the outputs from the scene modeling process as a single JSON file. After that, we produced an image of the frame using the blender camera. Every frame, we use Blender to erase the previous scene and build the scene from scratch. To streamline the process of drawing lanes with varying colors and types, we employed blender's geometry nodes. The lane geometry node draws the appropriate lane after receiving the lane type and cubic bezier control points.

III. Models Used for Different Cases

A. Lane detection

Masked RCNN, a model that predicts types of lanes in image space on monocular camera pictures, we utilized this model for lane detection which segments type of lane solid or dashed. But it does not classify the color of the lane detected. So we don't have the colors for the lane. We also took the data for road lane signs like right turn, straight arrow from this model. For solid line we took 5 points on the line and, for dotted line we took the starting and end points for the line, for road signs we used the whole contour of the sign. However, this model cannot work properly when there are no lane lines on the road.

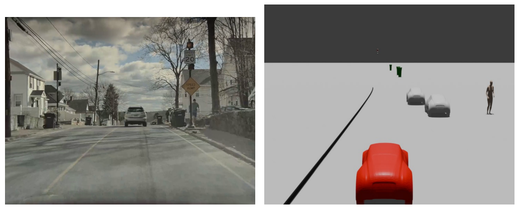


(a) Input (b) Output

Fig. 1. *Black Vehicles are Moving and White Vehicles are Parked*

B. Pedestrian Detection with Pose

The pedestrian detection was done by using OSX model. This model allowed for the direct retrieval of the 2D bounding box, mask, and confidence level for each detected pedestrian with their pose. To estimate the depth of the pedestrian, we employed Zoe Depth, a model capable of estimating depth from monocular images. To determine the position along the x and y axis we use depth for x-axis and calculated the center by converting the bounding box center in the world coordinates and then plot that center in y-direction, and then scaled the person accordingly.



(a) Input (b) Output

Fig. 2. *Input, Output Images*

C. Object Detection

For object detection and classification we used DETIC model. It consist of more than 20000 classes based on different dataset which gives output as the bounding box , mask and confidence of the detected object. We used this data to classify different types of vehicles, traffic light, cones, different types of traffic assets. All of them are detected using Detic model. We dumped all the data into a JSON file consisting of the labels , centers of the detected bounding box , the color of the traffic light if detected and again using Zoe Depth we calculated the depth of the object. To detect the color of the traffic light we extracted patch of the traffic light from detic output and used the classical approach for the detection of color in that approach we initially threshold the image to get mask for red, yellow and green color and then calculated the percentage of pixels in each masks, whichever percentage was higher we selected that as the color of the traffic light and if not detected we assumed the color to be red for safety reasons.

We tried to find some better model which detects the arrows in the traffic light but unfortunately we couldn't find any better pretrained model to detect the arrows of the traffic lights.

We were also not able to get the speed limit signs. The main drawback of this model was in its accuracy. It was able to predict even small objects from the image but the classification was not that quite accurate. In many cases, for example, it predicted SUV's as Truck.

D. Types Of Vehicles

For the classification of type of vehicle we used a repository Car-Model-Classification which is a better version of a published kernel on kaggle which was on ResNet34 architecture and the pretrianed model we used was trained using MobileNetV2 architecture. It was trained on stanford cars dataset. The output of this model gives us 49 different car company and 18 different car types, we made a small change in the code as we only needed the car type without the model of the car, once the car is classified we took the type of the car and if the car is not recognized in any type we assumed it to be a sedan.

E. Orientation of vehicles

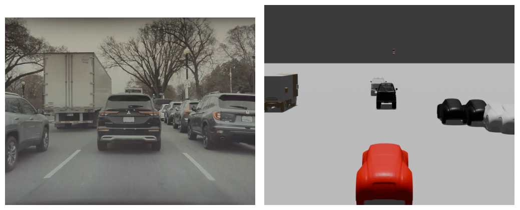
To determine the orientation of the vehicles we implement pretrianed yolo 3d to get the 3d bounding box of the car and from that we got the rotation as alpha and omega and rotated the vehicle based on the angle we got from the model. To make the model work based on our needs we had to modify the projection matrix with the intrinsic matrix of our camera and the translation we assumed it to be 0. YOLO3D was not that accurate in predicting the orientation of the vehicles. It was only able to predict orientation when there was a huge change in the orientation.

F. Break Light and Indicator

The image is loaded, and a car patch is extracted from the image based on the bounding box dimensions provided by the Detic. This region is converted to the HSV color space, and a mask is created based on pixel intensity. After applying erosion and dilation operations to refine the mask, contours are found within the filtered region. These contours are drawn on the original image, and their positions are adjusted relative to the original image. By analyzing the distribution of contours on either side of the region, the function classifies if brake lights, or left indicators or right indicators are on or off. This approach was not that accurate, it was only able to rightly predict the break lights and indicators 50% of the times.

G. Optical Flow

Opticalflow is computed between two consecutive frames of a video using the Farneback method provided by OpenCV. The function takes two frames (frame1 and frame2) and a point of interest (point) as input. It first converts the frames to grayscale and initializes an array for storing the computed flow. It then calculates the optical flow using *cv2.calcOpticalFlowFarneback*, converts the flow vectors to magnitude and angle representations, and normalizes them. Afterward, it computes the displacement of the specified point in the optical flow field and determines whether the object at that point is moving or parked based on a predefined threshold value. If the flow value is less than 2 (trial and error), the object is classified as 'Moving'; otherwise, it is classified as 'Parked'.



(a) Input (b) Output

Fig. 3. *Black Vehicles are Moving and White Vehicles are Parked*

References

[1] X. Zhou, R. Girdhar, A. Joulin, P. Kr¨ahenb¨uhl, I. Misra, Meta AI, The University of Texas at Austin, *Detecting Twenty-thousand Classes using Image-level Supervision* 2022.

[2] LANE DETECTION: Mask RCNN (https://debuggercafe.com/lane-detection-using-mask-rcnn/)

[3] DEPTH ESTIMATION: ZoeDepth (https://github.com/isl-org/ZoeDepth)

[4] HUMAN POSE: OSX (https://github.com/IDEA-Research/OSX)

[5] Optical Flow (https://stackoverflow.com/questions/38131822/what-is-output-from-opencvs-dense-optical-flow-farneback-function-how-can-th)

[6] Vehicle Classifier (https://github.com/kamwoh/Car-Model-Classification)