P3- Einstein Vision

Uday Sankar
usankar@wpi.edu
Using 1 late day

Gowri Shankar Sai Manikandan
gmanikandan@wpi.edu
Using 1 late day

Shaurya Parashar
sparashar@wpi.edu
Using 1 late day

***Abstract*—In this project,we aim to build a visualization system inspired by Tesla's latest dashboard for autonomous mode. The system takes a set of videos recorded from cameras of a 2023 Tesla Model S and outputs a rendered video of visualizations, including identifying lanes, vehicles, pedestrians, traffic lights, and road signs. In Checkpoint 1, the focus is on implementing basic features, while Checkpoint 2 adds more granularity to the system by sub-classifying vehicles, classifying arrows on traffic lights, and indicating objects such as dustbins and traffic poles. Checkpoint 3 adds further cognitive abilities to aid in decision-making, such as identifying brake lights and indicators of other vehicles, pedestrian pose, and distinguishing between parked and moving vehicles.**

I. Checkpoint 1

A. Lanes

We have implemented two models to get segmented lanes. Figure 1. is a sample output from LaneNet Lane Detection [7]. The YoloPv2 (Figure 2) [5] unsegmented clean lane points, whereas the working LaneNet model gives segmented yet unclean points. So we use the segmentation from LaneNet to match the clean lane points from YoloPv2, and get correct segmentation between lanes.

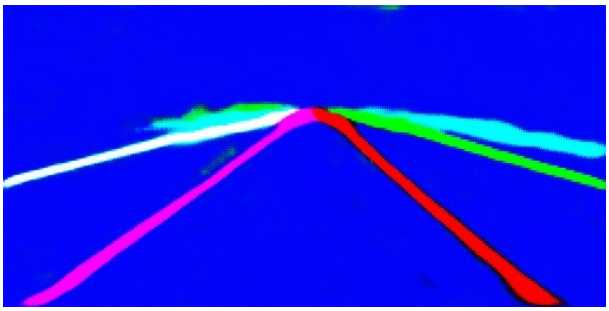

Fig. 1: LaneNet Segmented Mask

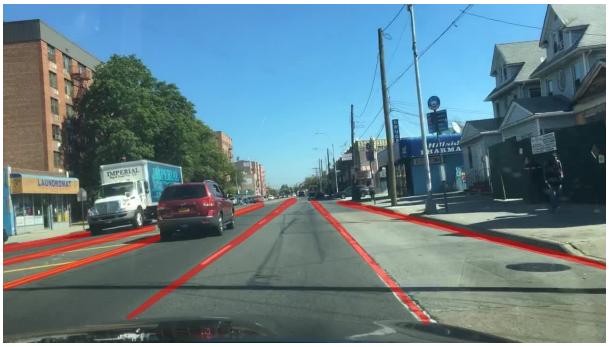

Fig. 2: YoloPv2 Sample Output

B. Vehicles, Pedestrian, Traffic Light, Road Signs

Yolo V3 (Darknet) [2] was used in order to detect the vehicles, pedestrian, Traffic Light, and Road Signs (Stop signs). In order to get the traffic light color, we made masks for both green and red color in HSV space. Passing them through the detected 2D bounding box, and counting the number of green v/s red pixels was our idea behind it. However, the intensity of the masks still remain a hyperparameter to get better and better accuracy. Figure 3 is a sample output of Yolo V3 detecting vehicles and Traffic Lights.

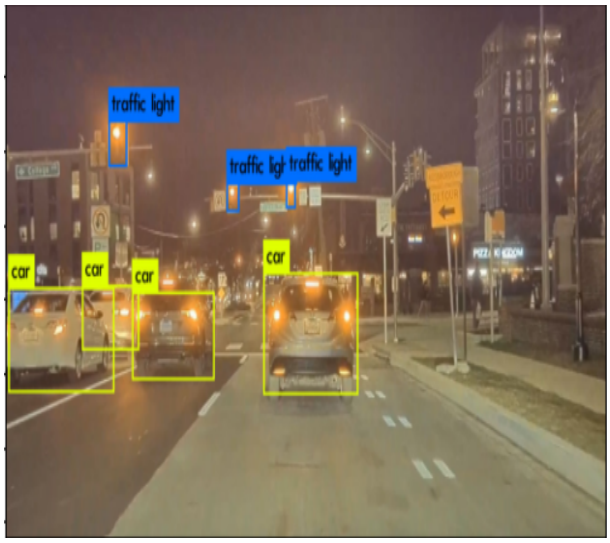

Fig. 3: Yolo v3 Detecting Vehicles and Traffic Lights

The flowchart (Figure 4) gives an overall pipeline of the models that we implemented, starting from the conversion of frames to sequences, to getting the depth map for the frames, and getting rendered sequences.

II. Checkpoint 1

A. MiDaS Depth Map Generator

For every single frame, we get depth maps from MiDaS, using which we spawn our objects in the environment, given the center points of the detecting boxes of the objects.

B. Visualization in Blender

Some of the visualization that we got from the detected objects for Checkpoints are included in the following figures.

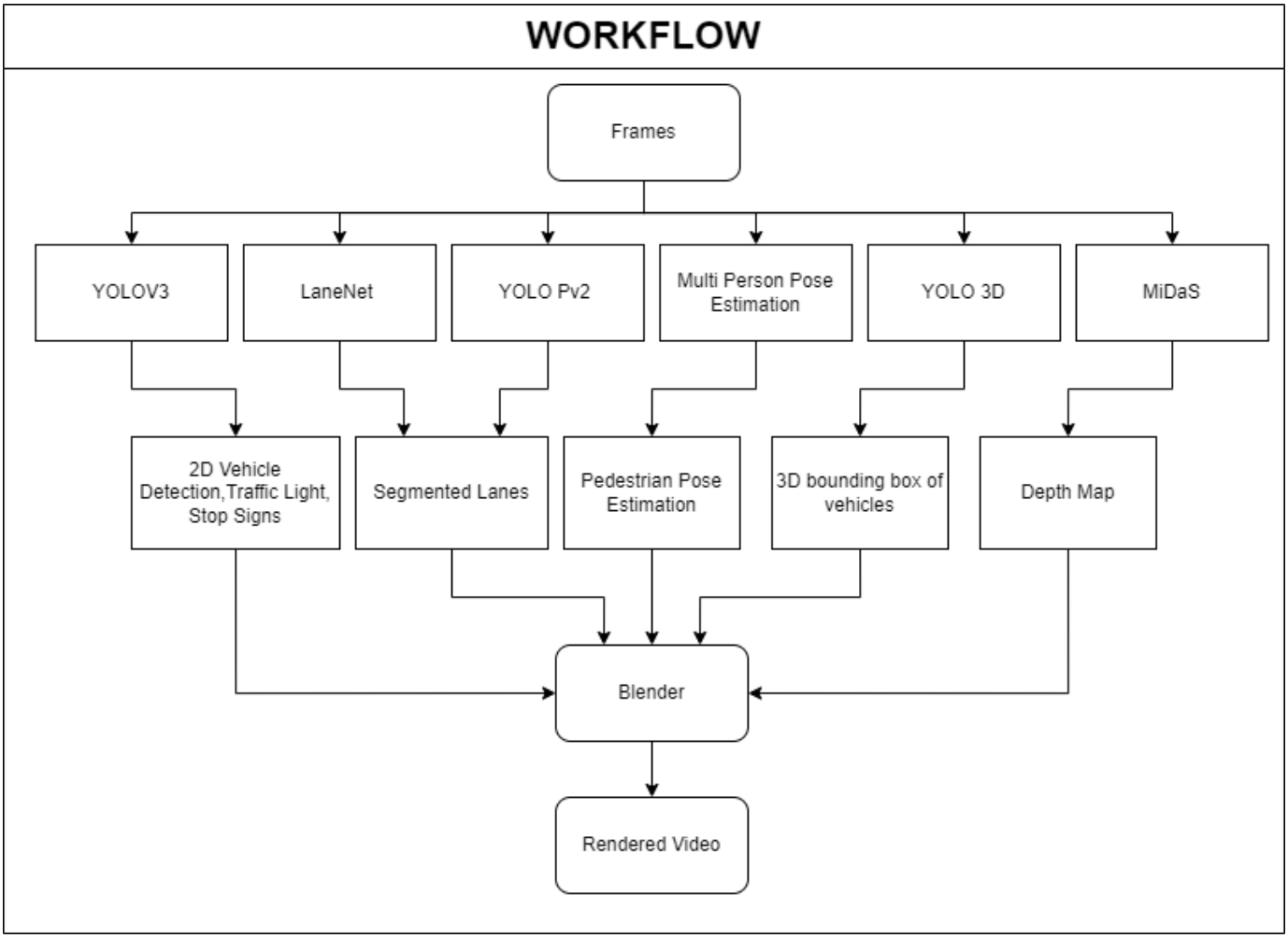

Fig. 4: The Model Zoo

Fig. 5: Visualization Sample 1- Checkpoint 1

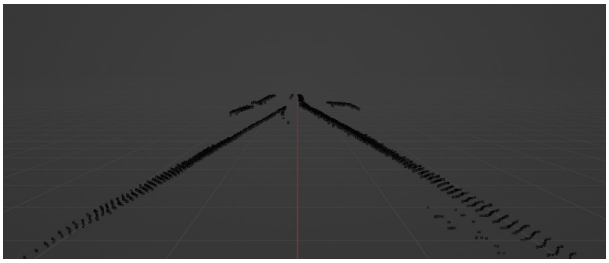

Fig. 6: Visualization Sample 2- Checkpoint 1

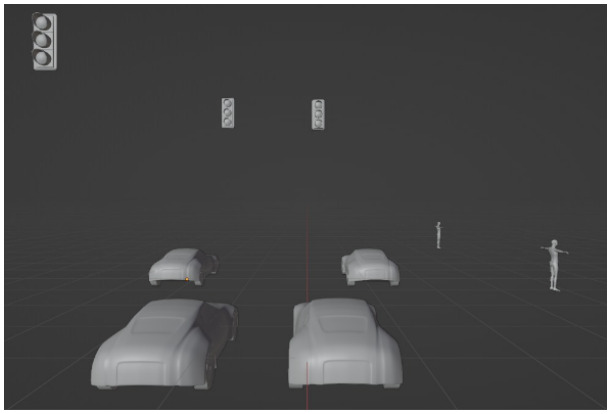

Fig. 7: Visualization Sample 3- Checkpoint 1

C. Challenges Faced in Checkpoint 1

1. Tuning the color range for thresholding the traffic lights, in order to segregate them into red and green light is a difficult task, to generalize for all the frames, be it night or noon.
2. Segmentation of the lanes are performed mostly by very poorly trained good models, or highly software incompatible models. For example- We found a LaneNet Tensorflow model

[4], the best trained model. But having setup on Tensorflow 1, we were unable to run it on either Colab, or on Local due to several dependencies being shut down on Tf1. Though Kaggle Notebook supports Tf1, due to several deprecated dependencies, it was hard to go through it.

III. Checkpoint 2

A. Vehicles

1. From the YOLO 2D detection of vehicles that we did in First Checkpoint, we wanted to shift to 3D bounding box, i.e. 6D Pose Estimation of Vehicles. We wanted to do this is because of the basic fact that given vehicle coordinates u,v, cannot give us the orientation of the vehicle to visualize in Blender . In specific, we are only concerned for yaw of the vehicles, because in the most extraordinary condition roll or pitch is considered.

2. We considered Baidu's Vehicle Angle Detection Competition [1]. We carefully studied and tried to implement the codes of the winners and runners up, which was basically based on mask CNN, and 3D bounding boxes respectively. However, despite repeated trials, it couldn't work unfortunately on our data. From this, we took inspiration to apply 3D bounding boxes.

3. YOLO3D [6] (Figure 8) was tried and implemented to find the image coordinates and yaw value. Initially we tried to implement both 2D and 3D object detection together, as YOLO3D gives very much awry values for location compared to 2D Detection, i.e. use 2D detection for vehicle classification and location, and yaw for angle. However, due to irregularities between vehicles being detected between both the models, which we tried to solve by looping over all the detected objects of a frame, and matching with nearest vehicle of 2D to 3D, we couldn't do it as planned as in a packed vehicle environment, it gets tough to match in that way.

Fig. 8: Sample YOLO3D

In the above image, it is clearly visible that 6D bounding boxes are not at all plotted correctly. It may be due to the fact that YOLO3D was trained using stereo camera, and inference was being done on Monocular images. So, maybe stereo camera properties might be coming into play somewhere during testing. Also, it might be the case that it is not trained enough to run well on unseen data.

However, we still continued with this 6D pose estimation, because we considered knowing angle of a vehicle to be more important than proper position coordinates of it, because position can be compensated by using scaling and shifting in the environment, due to the lack of metric scale. Figure 9 shows the vehicles spawned in their original orientation.

Fig. 9: Vehicles facing opposite in direction

B. Vehicle Classification

1. We tried different parameter comparisons from the bounding boxes in order to do vehicle classification. Like mentioned earlier, we had to scrap off the idea of doing 2D classification due to mismatch in detected objects in 3D and 2D models. So, we tried several ratios like the most easily visible ones- height:length, height:width, height:depth for classification. As they are hyper parameters, which we were only tuning for one frame, we think a more generalized approach like taking a set of values and check over all the images for the accuracy of classification should be a way better method. Our tuning for one frame was giving not so great accuracy for other frames.

C. Human Pose Estimation

We implemented the Human Pose Estimation model, Real time Multi-Person Pose Estimation [3]. The Figure 10, gave us satisfactory enough results to pursue with this. Although we tried to reason whether only u,v of human body checkpoints are needed to find the pose of a pedestrian, we came to a conclusion that the depth values of different keypoints will be able to place the body armature (Figure 11) such that pose of a pedestrian is correctly visualized.

Fig. 10: Sample YOLO3D

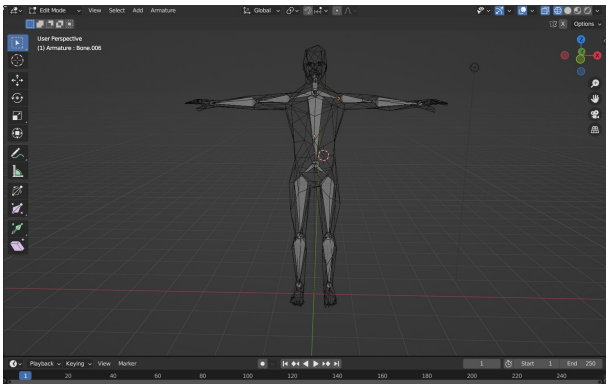

Fig. 11: Sample YOLO3D

D. Lanes

Plotting a mesh containing the lane points as we did for Checkpoint 1 is clearly not very pleasing to the eye and would generally not be considered a good visualization. In an attempt to improve this, we decided to plot the lanes as profile curves in Blender.

The main issue with using lane segmentation with blender visualizations is that the models provide a list of pixels. The large number of pixels we get is a lot of information. In order to draw neat lanes in Blender we would need only a tiny amount of this data, but it needs to be high quality. Therefore, it is the selection of good points that makes it harder than other tasks in the project.

In our implementation, we first combine the instance segmentation information from our poorly trained Lanenet model. We took the data points from a YOLOpv2 as they were cleaner, and used an intersection of points from both networks. We first project these points from the image to the 3D world points on the road and then fit a second-degree polynomial curve that minimizes the sum of the L2 distance of the points from the curve. This is a simple regression problem for which we used RANSAC. After we get the curve, we sample points on this curve. These sampled points act as control points for a bezier curve that we draw in Blender. We then fit a profile to this bezier curve which makes it look like a lane marking.

We have implemented this method for one frame, but our hyperparameters need more tuning. We plan to generate video sequences for these lanes in the next few days.

IV. References

References

[1] https://www.kaggle.com/competitions/pku-autonomous-driving/leaderboardKaggle, 2019. Available online: https://www.kaggle.com/competitions/pku-autonomous-driving/leaderboard.

[2] Alexey AB. Darknet. https://github.com/pjreddie/darknetGitHub Repository, 2022. Available online: https://github.com/pjreddie/darknet.

[3] Zhe C. Realtime multi-person pose estimation. https://github.com/ZheC/RealtimeMulti-PersonPoseEstimationGitHub Repository, 2017. Available online: https://github.com/ZheC/RealtimeMulti-PersonPoseEstimation.

[4] MaybeShewill CV. Lanenet-lane-detection. https://github.com/MaybeShewill-CV/lanenet-lane-detectionGitHub Repository, 2022. Available online: https://github.com/MaybeShewill-CV/lanenet-lane-detection.

[5] fing fs. Yolopv2. https://github.com/CAIC-AD/YOLOPv2GitHub Repository, 2022. Available online: https://github.com/CAIC-AD/YOLOPv2.

[6] Didi Ruhyadi. Yolo3d. https://github.com/ruhyadi/YOLO3DKaggle, 2022. Available online: https://github.com/ruhyadi/YOLO3D.

[7] Iroh Xu. Lanenet lane detection pytorch. https://github.com/IrohXu/lanenet-lane-detection-pytorchGitHub Repository, 2021. Available online: https://github.com/IrohXu/lanenet-lane-detection-pytorch.