RBE 549 Project 3
Einstein Vision

Used two late days for Project 3

Niranjan Kumar Ilampooranan
MS Robotics Graduate Student
Worcester Polytechnic Institute

Thanikai Adhithiyan Shanmugam
MS Robotics Graduate Student
Worcester Polytechnic Institute

I. Introduction

User interface is a key component in systems that heavily rely on interaction between the user and the robot. Better UI directly translates to seamless interactions and so, it is crucial for the interface to be intuitive for ease of use and understanding. In the case of self-driving cars, an UI that can provide valuable insights on the scene without clutter can go a long way.

For this project, an attempt is made to recreate the information capture (dashcam footage) of various objects typically seen on the road and provide a neat visualization of said objects. This would include lane markings, pedestrians, and vehicles in the proximity. In further sections, key concepts that were implemented for this project is elucidated which include the following -

- Estimating the location of objects
- Detecting lane markings
- Estimating Pose of pedestrians
- Rendering the entire scene captured on the camera

Results, notable observations, and encountered pitfalls are discussed in the relevant sections.

II. Object Detection and Pose Estimation

Object detection refers to the capability of computer and software systems to locate objects in an image/scene and identify each object. We utilised several models and datasets which helped us detect a variety of objects in outdoor environments. [1]

We started Phase 1 by inferring the widely used YOLO Model version 9 by ultralytics with the pretrained weights from the MSCOCO Dataset. The YOLO model uses the Darknet Architecture and generates the bounding boxes of the detected objects. YOLO inference high mAP for cars, motorcycles, trucks, stop signs and pedestrians. However, the other objects mentioned were performing poorly or were not detected by YOLO. To improve the efficiency in object detection, we tested our data on the Detic Dataset. The Detic Dataset was a object-detection model by Facebook Research and can detect 30,000 classes. The retained weights we tested on was trained on the MSCOCO, Object36 and OpenImages datasets. The model was able to detect traffic cones, speed limits, trash bins. Detic (A Detector with image classes) was also able to distinguish sedans, SUVs ,Trucks and minivans. However, in terms of efficiency, we noticed that YOLO performed detection better in vehicles and pedestrians compared to Detic (classification result shown in figure below). Therefore, We tested YOLO to generate vehicles and resonated with Detic to generate the other objects and distinguished the type of vehicle by masking the bounding box detected by YOLO over the Detic[2] data.

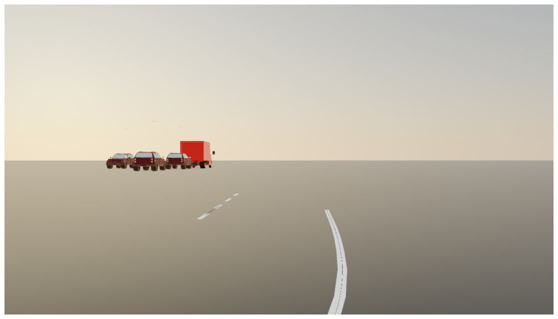
Fig. 1. Classification result of cars - SUV using Detic

Once the object is detected, the next milestone lied in extracting the pose of the object from the image to project it in the 3D image. To achieve this, we deducted some assumptions based on observations that there is no roll, pitch, yaw for other objects apart from vehicle and pedestrian. and no roll, pitch for vehicle(4DOF).

Based on these assumptions, we utilised the YOLO3D[3] model for this purpose. YOLO3D uses 2 different models to generate the 3D pose of vehicles. YOLO3D starts with detecting the 2D bounding box of vehicles using YOLOv9. Then, the model uses a regressor network which predicts the 3D bounding box by regressing the 3D properties and comparing with constraining the properties with the 2D box. Using the pose of the 3D pose, we extract the yaw of each vehicle/bounding box.

The YOLO3D model has problems with efficiencies as the 3D bounding box detected is too low. We tried various 6D pose detection models but YOLO3D was by far the best. The pretrained weights were trained on KITTI dataset for 25 iterations which was pretty low for a huge dataset.

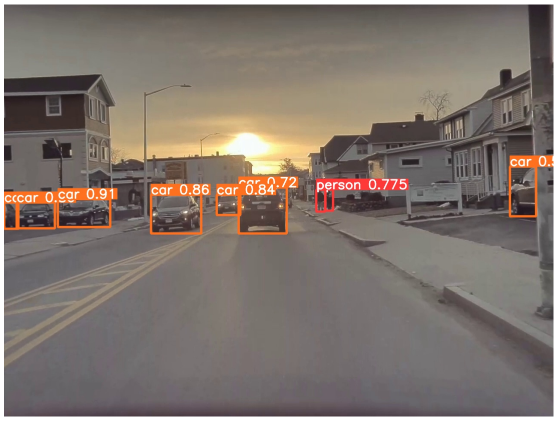

Fig. 2. YOLO Detection

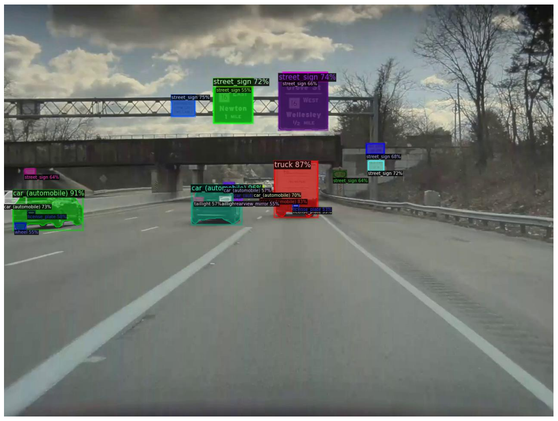

Fig. 3. Detic Detection

III. Depth Estimation

The crucial information missing in an image which reduces the knowledge on world environment from image is depth. Depth can be calculated in terms of relative and metric scale. We focused in this phase towards models which estimates metric/absolute depth. We started with the widely used MIDAS dataset in Phase 1 which provide the relative depth of the image. However, MIDAS was efficient for smaller distance but in outdoor dataset like ours, it was not able to predict depth of any object in the image. Considering very poor performance of MIDAS[4], we switched to Zoedepth.

Unlike Midas, it gives us metric depth and had better accuracy than MIDAS. For our, we were able to partially detect the data of the predicted objects in the dataset but the whole object resolution was poor. Also, Zoedepth[5] has the problem of scale ambiguity as the model is unable to predict the scaling factor if the object are far away and assumes that the whole image is further than the estimated depth. Also, since Zoedepth is also trained on indoor environments, it produces undesirable outputs at various frames. In our project, we saw that the car was teleport to further and nearer distances simultaneously with the dataset.

Therefore, the final alternative we selected was the Marigold[6] model. The Marigold is a diffuser model which is trained on a large number of images. Marigold is derived from Stable Diffusion and fine-tuned with synthetic data, can zero-shot transfer to unseen data, offering state-of-the-art monocular depth estimation results. Marigold also provides with absolute depth. The marigold outputs we generated were far refined and accurate than zoedepth. Previously, Lane data with zoe depth resulted in curved lane even if the predicted bounding box was straight. We used depth from Marigold to compare with pixel coordinates of the centroid of the bounding box to project it in the blender environment.

IV. Lane Detection

Lane detection and segmentation have a lot of use cases, especially in self-driving vehicles. With lane detection and segmentation, the vehicle gets to see different types of lanes. Solving lane detection often proves challenging even with large datasets. For solving this, we utilise the Mask R-CNN model [7] to generate bounding boxes for the detected lanes.

The model detects the entire solid lane and dividers as a single 2D bounding box while for dotted lines, it detects as independent smaller bounding boxes and merges them together into a single array. The pipeline then uses Bezier curve to fit the line as we know the starting and end points of the line(corners of bounding box). The Mask RCNN identifies solid lines and dotted lines with ease but faces difficulty in identifying double lines and dividers. Also, the model is not able to distinguish the color of the lines. The Bezier curve fit is not accurate in terms of fitting. We planned to explore spline interpolation to fit the curve but due to time constraint we settled for Bezier curve.

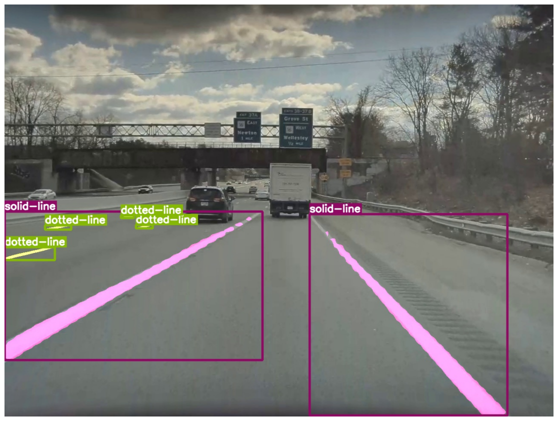

Fig. 4. Lane Detection using Mask R-CNN

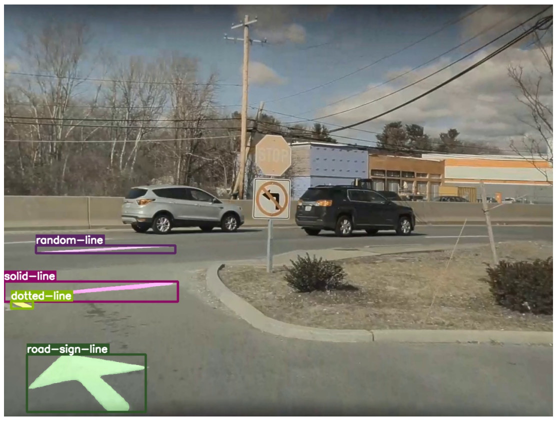

Fig. 5. Arrow Detection using Mask R-CNN

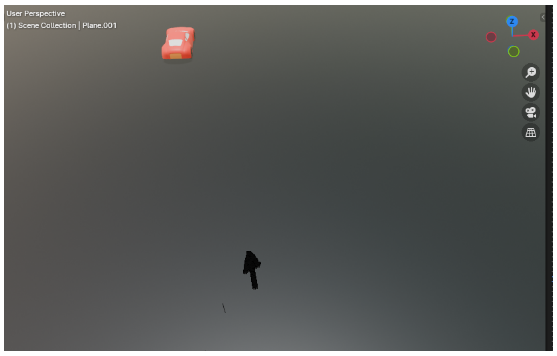

Fig. 6. Arrow marking plotted in Blender

V. Traffic Light

Detecting Traffic Light proved to challenging than we anticipated. We began by inferring the YOLOv8 model with the a custom dataset.[7] The model works reasonably well when the traffic lights are placed at a close distance but was not able to detect from further distances. Therefore, we had to create a mask and use classical approach to identify light. We traffic light is detected by converting the image to HSV and then creating a mask and thresholding the brightness to get the color of the traffic light. The classical approach was accurate at any distance and even for small traffic lights.

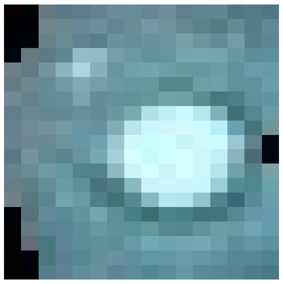

Fig. 7. Light Classification - Classical Approach

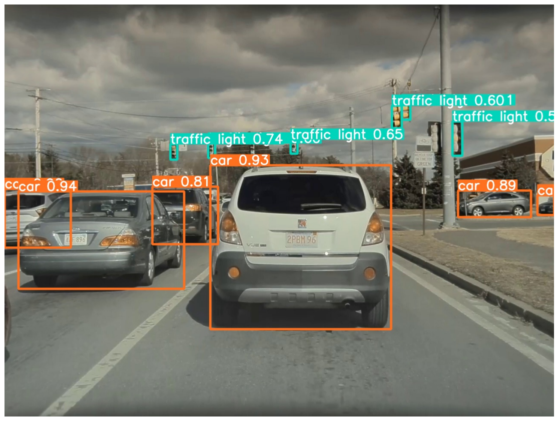

Fig. 8. Traffic Light Detection -YOLO

VI. Optical Flow

For this project, it is important to estimate the static and dynamic objects in the environment in order for smooth navigation and to avoid collision. We solve this problem using the optical flow. We find the flow in the image by using aa optical flow model and estimate the direction and magnitude of velocity of the object in the image.

We start by testing the RAFT model[8] which estimates the optical flow in the x and y directions and outputs a flow image. Meanwhile, the Fundamental Matrix F is estimated by computing the SIFT featues and extracting the corresponding points. Once the correspondences are detected, the points on image1 are reporjected to image2 and the Sampson distance[9] is computed between the correspondences and the reprojected points. Once, the distance is computed, the error is calculated by the difference magnitude of the flow vectors and the distance. Thresholding is done to check whether the object is moving or static with the error computer computed.

We represent the stationary objects especially cars, SUVS and trucks as green and the moving objects as red. We also indicate the predicted motion of the car by displaying an arrow on the car overhead. We faced couple of challenges while predicting the movement. The major issue was determining the threshold for each scene as the each scene is displayed with different properties and threshold changes accordingly. We planned on implementing variable thresholding by mean average error but could not implement due to insufficient time.

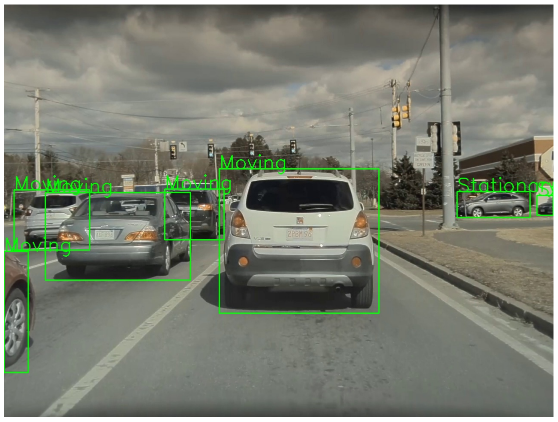

Fig. 9. Optical Flow using RAFT

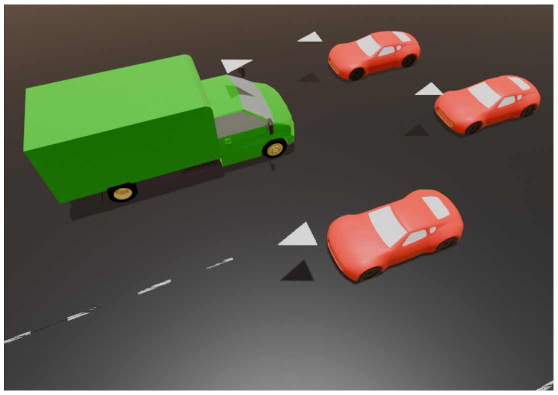

Fig. 10. Arrows indicating heading of vehicles

VII. Human Pose Estimation

Several methods exist in literature to estimate human pose (be it 2D or 3D). OpenPose can be used to extract 2D pose of the human but it is insufficient to create an armature that would resemble the pose of the pedestrian. In other words, a 3D pose of the pedestrian needs to be estimated, either through set of frames or using just a single frame. One such method that was employed for this project was I2L-MeshNet by Moon et al. [10].

I2L-MeshNet, in a nutshell, is a heatmap based 3D pose and mesh predictor that generates a mesh file (.obj file that can be imported in Blender) of the human detected using a single frame. It used PoseNet and MeshNet in a cascaded manner to estimate - a) lixel-based (line + pixel) 1D heatmap of the human joints in the image. The joint information serves as an important geometric feature that can provide important information on the mesh vertex locations. This along with the image features are fed to the MeshNet to predict the heatmaps of each 3D human mesh vertex coordinates to generate the .obj file. A sample output is shown in the figure below.

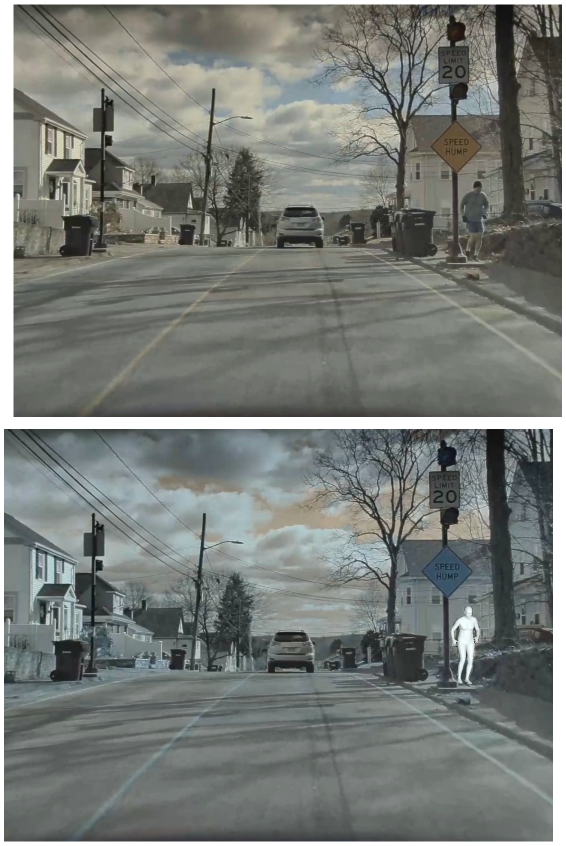

Fig. 11. Jogging Pedestrian and Rendered Mesh with Estimated Pose

As input to the model, the frame along with the bounding box of the human is provided. This bounding box can be obtained from object detection models like YOLOv8. The generated mesh file is moved to the Blender object directory and the process repeats for every frame (the same frame multiple times if there are more humans).

VIII. Scene Rendering

Blender is a versatile computer graphics software tool that provide the means to render the 2D dashcam footage for a sleek visualization in 3D. Everything put together from the output of various models discussed above, is then fed to Blender as a single file. Object models for each of the detected objects to be visualized were already provided.

A basic visualization of the simple features on the scene submitted for Phase 1 of the project is shown below. For this phase, emphasis was placed on the placement of the detected objects (vehicles, pedestrians, etc). Given the pixel coordinates of the centroid of each object, the camera extrinsics and intrinsics matrices are required to estimate the world coordinates of the same point. The camera intrinsics data was provided and the extrinsics was assumed (in our case, no translation and a small pitch angle of $10°$. And so, the world coordinates were identified and the objects were plotted on the $\mathbf{Z} = \mathbf{0}$ plane. Color changes are also represented (in the case of traffic lights and indicator lights) by change of material color of the respective segments.

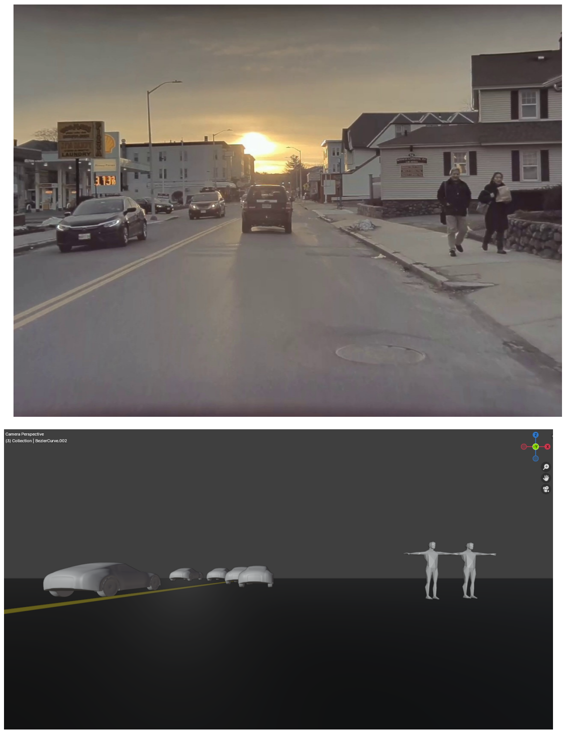
Fig. 12. Phase 1 output - Captured Footage vs Rendered Scene

For the second phase, few other features were also represented such as extra traffic assets (speed limit signs) and pose of detected pedestrians. Improvement from Phase 1 to Phase 2 in Blender representation was achieved by colouring the relevant models and scaling down the objects to their real-world sizes (the provided models were scaled up).

For both Phase 1 and Phase 2, lanes were plotted using Bezier curves. With the points from the lane detection model as the control points, there are methods to plot curves in Blender, of which NURBS and Bezier are prevalent in use. With the handles set as automatic to generate curves as smooth where possible, a variety of lane markings can be plotted such as dotted lines with a constant offset in the array of plane meshes and solid lines that closely mimic the curve. In the figure below, the solid lane markings, extra assets (dustbins and speed limit signs), and pedestrian pose (shifted from the default T-pose) is shown.

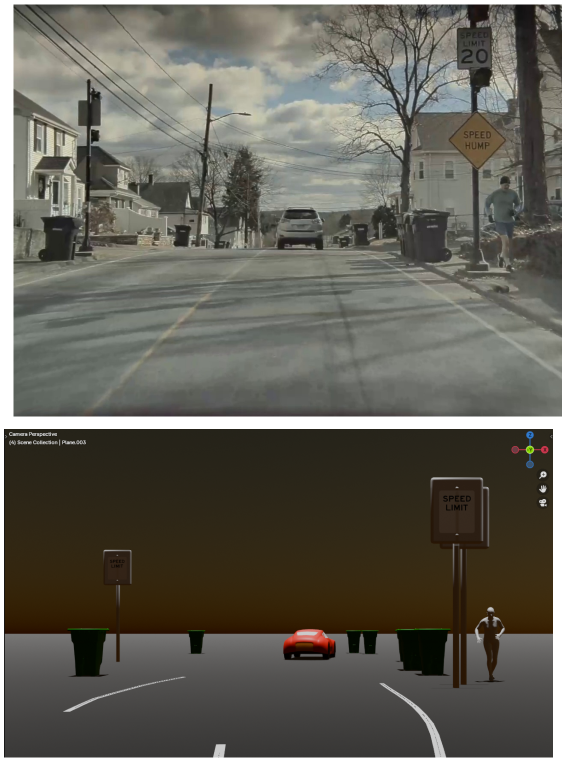

Fig. 13. Phase 2 output - Captured Footage vs Rendered Scene

As part of Phase 3, optical flow was implemented in which moving vehicles in the frame of the car are represented in red and stationary ones are represented as green. The video was rendered for all 13 scenes with varying results. Depending on the time of footage shot, the sky texture was modified accordingly.

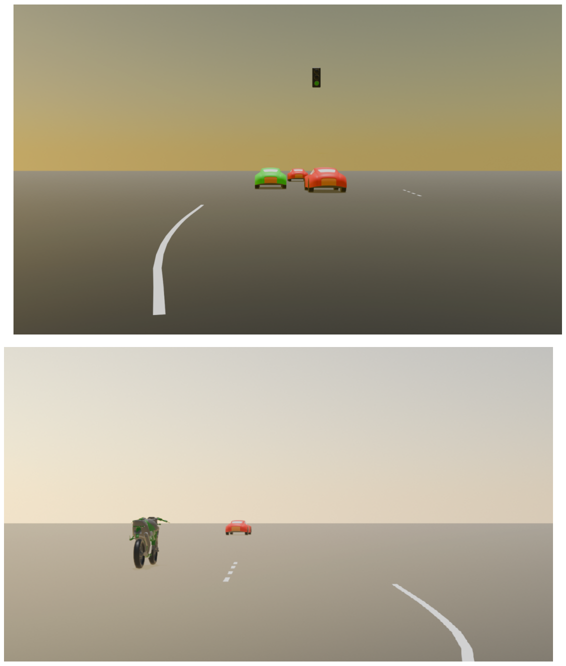
Fig. 14. Phase 3 output

Some of the pitfalls encountered during one-to-one representation in Blender include partial detection of vehicles when occluded. As it can be seen in the figure below, the bounding box for the car behind the wall partially coincides with the wall in front. This results in consideration of the depth data of that pixel, which places the car right in front of the jogging pedestrian (it is not the case in the footage). One possible solution would be tracking the non-occluded part of the object to extract the correct depth information. Another problem lies in the case of height placement - cars on bridge or two signs attached to the same pole.

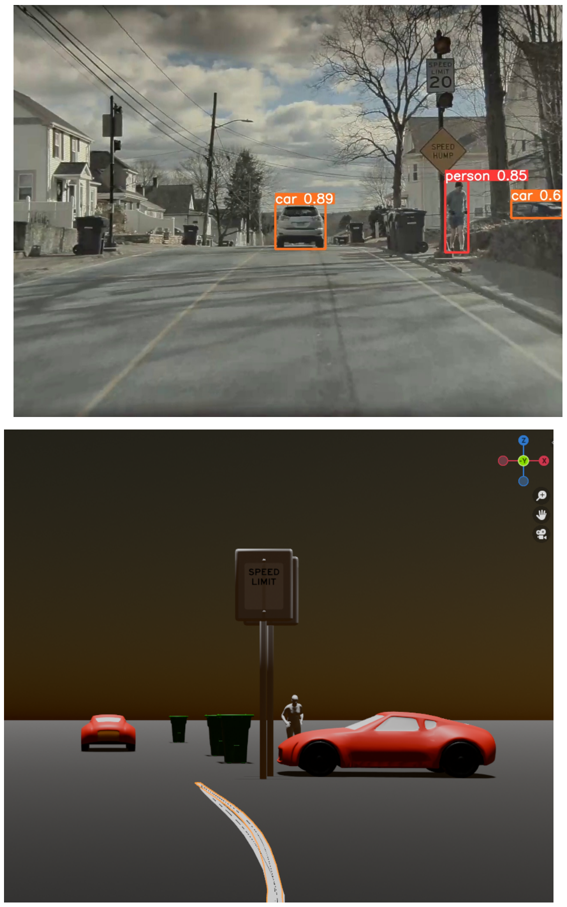

Fig. 15. Blender Object placement - Occlusion Pitfall

Additional corner cases include when only a part of the pedestrian is captured in the footage. For example, in the figure below, the pedestrian is almost out of the frame but is detected. In this case, only a small part of information is available for the I2L-MeshNet to work with. This results in generation of awkward poses that does not even remotely look like the pedestrian in the frame. One solution would be excluding the pedestrians in the corner since they do not provide valuable information in any case or augmenting with information from other cameras to estimate proper pose. In the likelihood of only the upper part of the pedestrian captured in the footage, similar bad results can be expected (see figure below). This also applies to cases where the person is difficult to distinguish from background.

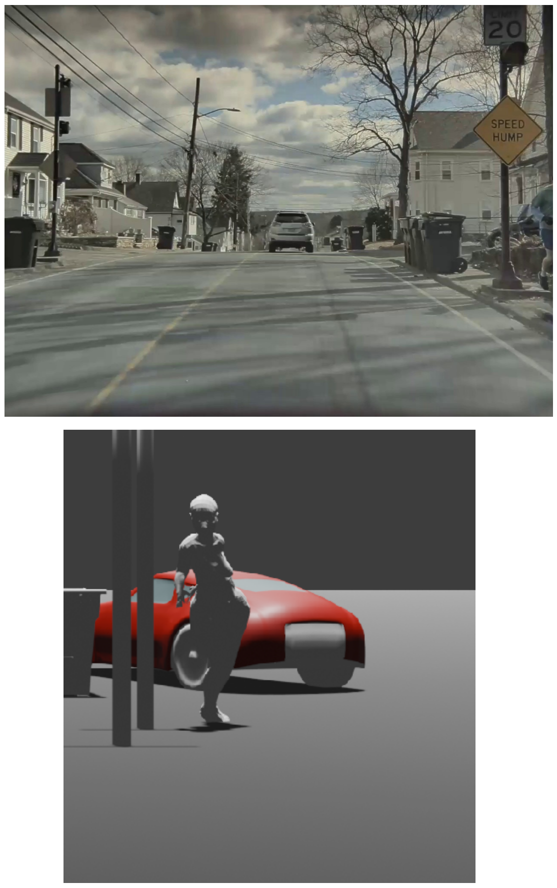

Fig. 16. Blender Object placement - Human Pose Pitfall

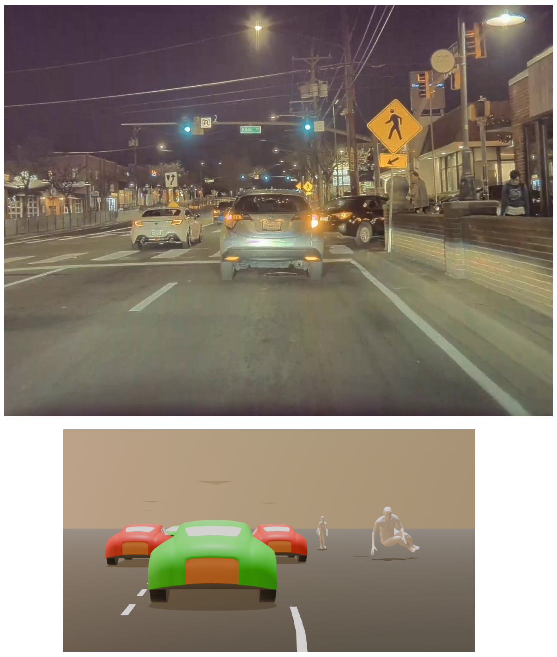

Fig. 17. Blender Object placement - Human Pose when only part of body visible

IX. Extra-Credit Speed Breaker Detection

The bonus part of the project included detection of speed breakers in the image and projection in the blender environment. To achieve this, we implemented the YOLO v7 model with the custom trained dataset [**?**] which detects the speed breakers as bounding box. YOLO provides with centroid and the width and height of the speed breaker. We infer the depth of the bounding box centroid using the Marigold data and project it on the environment. For creating a mesh. we create a cylinder which has diameter equal to to the height of the bounding box and length equal to the width. We then project half the cylinder above the ground while the remaining half gets buried under the ground.

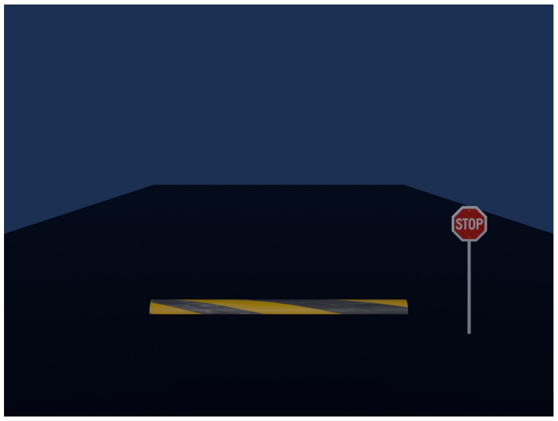

Fig. 18. Speed Breaker in Blender

References

[1] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "Yolov9: Learning what you want to learn using programmable gradient information," 2024.

[2] X. Zhou, R. Girdhar, A. Joulin, P. Krähenbühl, and I. Misra, "Detecting twenty-thousand classes using image-level supervision," in *ECCV*, 2022.

[3] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," 2017.

[4] R. Birkl, D. Wofk, and M. Müller, "Midas v3.1 – a model zoo for robust monocular relative depth estimation," *arXiv preprint arXiv:2307.14460*, 2023.

[5] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, "Zoedepth: Zero-shot transfer by combining relative and metric depth," *arXiv preprint arXiv:2302.12288*, 2023.

[6] B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt, and K. Schindler, "Repurposing diffusion-based image generators for monocular depth estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[7]

[8] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," 2020.

[9] H. Zhang and C. Ye, "Sampson distance: A new approach to improving visual-inertial odometry's accuracy," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9184–9189, 2021.

[10] G. Moon and K. M. Lee, "I2l-meshnet: Image-to-lixel prediction network for accurate 3d human pose and mesh estimation from a single rgb image," 2020.