RBE 549: Project3 - Einstein Vision (Using 2 late day)

Smit M Shah
Email: smshah1@wpi.edu
Worcester Polytechnic Institute

Rigved Sanku
Email: rsanku@wpi.edu
Worcester Polytechnic Institute

***Abstract*—In this project, we built a simple yet effective visual system for self-driving cars, inspired by Tesla's dashboard. We used a mix of new and classic methods from computer vision to solve common challenges in autonomous driving. Our approach focused on achieving a clear, intuitive representation of the vehicle's surroundings using Blender, ensuring both safety and ease of understanding for human operators. Through this integration, we aimed to enhance the interaction between humans and autonomous driving systems, providing a reliable and user-friendly interface.**

I. Introduction

In the realm of technology, particularly when it involves human-machine interaction, the power of effective visualization cannot be overstated. Good visualizations are not just about presenting data or information; they are about translating complex processes into intuitive insights. This is especially critical in the context of autonomous systems, where humans must rely on and interpret the decisions made by machines.

Tesla's latest dashboard visualization exemplifies how sophisticated, well-designed visual interfaces can significantly enhance HRI. By offering detailed and intuitive insights into the vehicle's perception and decision-making, it sets a benchmark for what effective visualization in autonomous driving should entail.

This project is motivated by the need for advanced visualization tools that not only address the technical requirements of autonomous vehicles but also prioritize the user experience. By integrating state-of-the-art deep learning and classical computer vision techniques, we developed a visualization system that bridges the gap between human intuition and autonomous machine logic, thereby enhancing user trust in autonomous vehicles.

A. Dataset Overview

The dataset comprises:

- A collection of 13 video sequences captured under diverse environmental settings. Each sequence is available in both its original format (raw) and a version corrected for lens distortion. (undistorted)
- A set of calibration videos intended for camera calibration purposes.
- A variety of 3D models (*Blender Assets*) including various vehicle types (Sedans, SUVs, Pickup Trucks, Bicycles, Motorcycles, and Trucks), as well as road infrastructure elements (Traffic Signals, Stop Signs, Traffic Cones, Traffic Poles, Speed Signs) and a model representing pedestrians. Additionally, texture files for the stop sign and a template for the speed sign are provided.

B. Model Pipeline

- **Frame Sampling:** Our process begins with the selective sampling of video data, where we extract every 10th frame from the given videos. This step ensures we work with a simplified dataset that retains essential temporal information while being computationally manageable. Figure 1 shows the Depth generated from unidep alongside the Original Image
- **Depth Estimation:** We utilize the UniDepth [1] model, renowned for its depth estimation capabilities, to analyze each chosen frame. This model provides depth information for every pixel, giving information about the relative positions of objects. Figure 2 shows the depth map of generated from Unidepth.

Fig. 1. Depth Estimation using Unidepth

- **Object Detection and Classification:** Our pipeline integrates a mix of pre-trained models, supplemented by retrained versions when necessary, to identify a wide range of elements within the frames. This includes detecting various vehicle types, pedestrians, and key urban infrastructure such as traffic lights and road signs. Additionally, it encompasses the identification of common roadside objects like dustbins and safety cones, which are integral to navigating complex urban environments.
- **Data Representation:** The detection phase yields classifications and the centroids of the bounding boxes encompassing the identified objects. We compile this information into a structured JSON file, serving as a detailed record of the detected elements within each frame.

- **Scene Reconstruction:** Using the pin-hole camera model and the calibration matrix details, we employ Python scripting within Blender to meticulously reconstruct the scene depicted in each frame. This involves initiating each frame's reconstruction from scratch, thereby ensuring the accuracy and freshness of the scene representation. The reconstructed scenes incorporate the camera's perspective, offering a realistic and immersive visualization of the vehicle's environment.
- **Rendering:** For each scene we reconstruct in Blender, we render a set of images that correspond to a video sequence. To compile these images into a continuous video, suitable for submission, we utilize online tools that stitch together the individual frames, creating a smooth animation.

Figure 2 shows the flow of the overall model pipeline

II. Checkpoint I

A. Lane Detection

In addressing the task of identifying various types of lanes—dashed, solid, and of different colors—on the road, we initially experimented with ClrNet [2]. However, we noticed that ClrNet predominantly excels in detecting mostly straight lanes, such as those found on highways or freeways, but falls short in recognizing the diversity of lane types.

Later, experiments were done using Mask RCNN [3], which proved to be better for lane detection. It was chosen due to its ability to detect and segment objects in an image with high precision and accuracy. It was able to identify and segment lanes in a variety of driving environments, including highways, urban streets, and rural roads.

Mask RCNN is able to perform the necessary instance segmentation, and is able to detect

- Divider-Line
- Dotted-Line
- Double-Line
- Random-Line
- Road-Sign-Line
- Solid-Line

Figure 3 shows an example output of the different types of lanes detected using Mask R-CNN.

To transform the 2D lane detections from Mask-RCNN into a 3D metric space, we employed a two-step assumption-based approach. Initially, we posited that the camera's position in the world frame is fixed at (0, 0, 1.2) meters. Our second hypothesis was that all lanes lie flat with no elevation, effectively having a height of zero. These assumptions allowed us to project the detected 2D lane pixels into 3D coordinates, setting the z-coordinate to 0 meters to reflect the lanes' flat nature. Following the successful mapping to 3D space, we applied a cubic bezier curve fitting to accurately model the lanes' shapes. The points used to fit the curve we used the first, the last and the middle 2 points of the segmented mask, enhancing the precision of our lane representation in the 3D space.

Figure 15 shows an example output of a set lanes fitted with bezier curves.

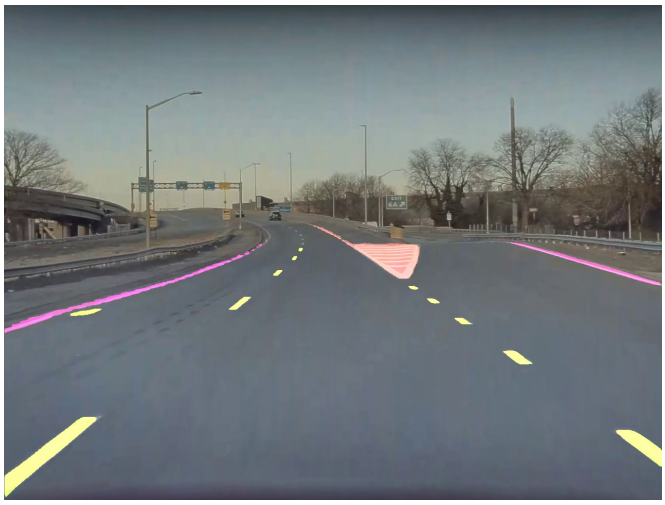

Fig. 3. Lane instance segmentation using Mask- RCNN

B. Vehicles, Pedestrian, Traffic Light, Road Signs

1) Vehicles:: For vehicle detection we used Yolo V8. We extract the centroid wrt the image frame of bounding boxes of the detected vehicles and then project these 2D Points to 3D points using the concept of similarity triangles, to spawn the corresponding vehicle in Blender.

2) Traffic Lights, Pedestrains and Road Signs: For Traffic Lights we use Detic [4]. Detic is cable to identify traffic signals as red, yellow, or green by adjusting the detection's confidence threshold levels. Additionally, it can recognize stop signs and pedestrians, providing a mask, bounding box, and a confidence score for each detection. Figure 4 shows the detection of all possible traffic lights at a junction road in the Night scene.

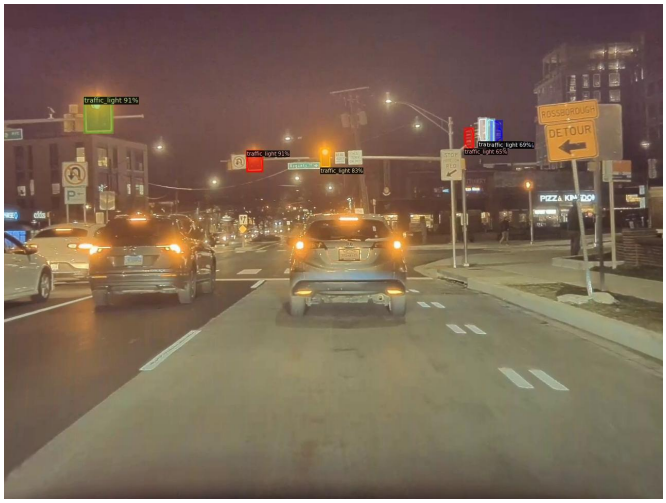

Fig. 4. Traffic Light Detection in Night Scene using Detic

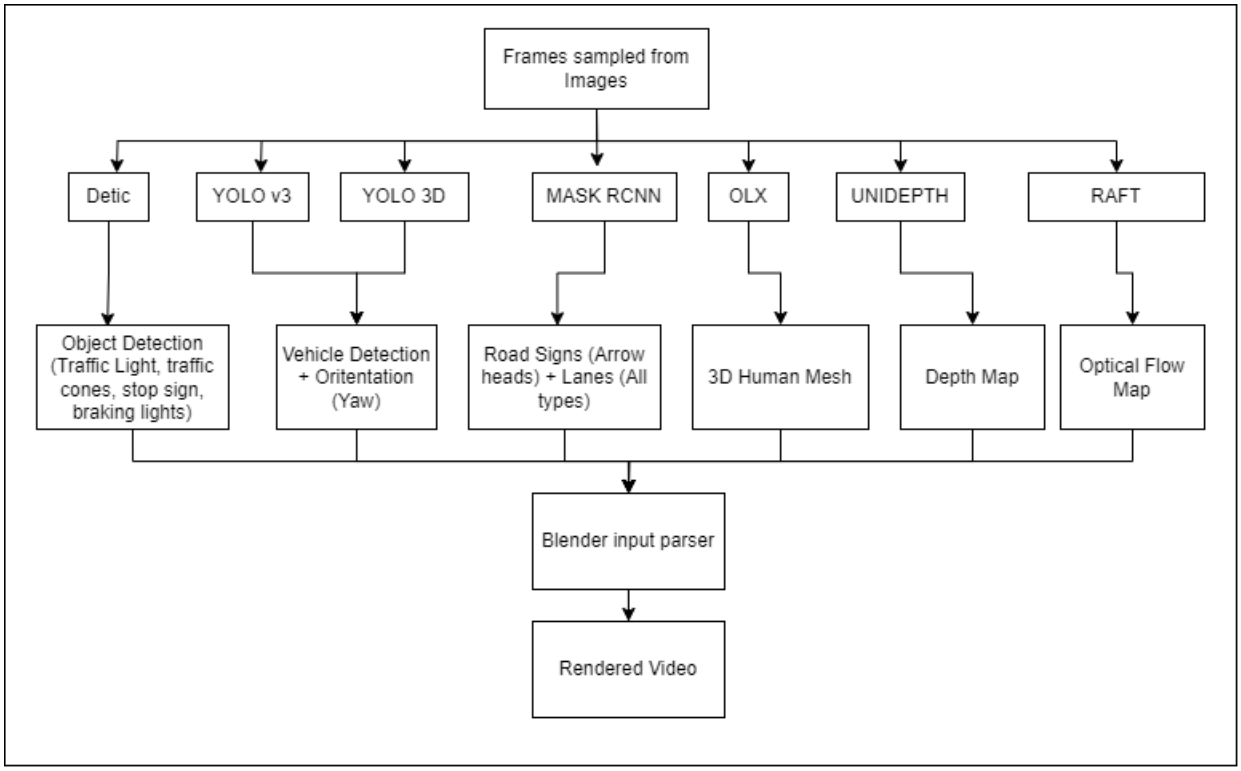

Fig. 2. Model Pipeline

III. Checkpoint II

A. Vehicles:

After working with 2D vehicle detection using YOLO in our first checkpoint, we decided to move towards 3D bounding boxes, or 3D Pose Estimation for vehicles (yaw, pitch, role). Mostly, we were interested in the yaw of the vehicles since roll or pitch changes are pretty rare.

We found a workable Github [5] that implements this 3D pose estimation paper [6]. This framework contains two networks. One for 2d bounding box detection. One for dimension and orientation estimation. With this information, the 3D pose and 3D bounding box can be estimated. We then tried using this YOLO3D to get the image coordinates and the yaw value. Initially, our plan was to combine 2D and 3D object detection – using Yolo v8 for 2D detection for pinpointing and classifying vehicles and YOLO3D for the yaw. But we ran into issues syncing up detections between the two models. In crowded scenes with many vehicles, matching the 2D and 3D detections proved to be particularly tricky. The inaccuracies in the 3D bounding boxes, as shown in the figure, might be because YOLO3D was trained with stereo camera data but tested on monocular images, which could cause discrepancies. It's also possible that the model wasn't fully adapted to handle new, unseen data.

Despite these hurdles, we stuck with 3D pose estimation, valuing the vehicle's orientation over its exact position. After all, we figured we could adjust the position by scaling and shifting within the environment, especially since we don't have a consistent metric scale to work with. As shown in the figure, this allowed us to place vehicles in their true orientations, enhancing the realism of our visualizations.

Possible Modifications:

1) **Adjustment of Projection Matrix:** An observed anomaly is the upward tilt of the estimated 3D bounding boxes, as opposed to a desired orientation parallel to the ground. A future adjustment might involve replacing the current projection matrix with the camera's intrinsic matrix to rectify the bounding boxes' alignment.
2) **Enhancing Bounding Box Precision:** The 3D bounding box accuracy is impacted by the presupposition that 2D bounding boxes closely envelop the object. The often-imprecise 2D bounding boxes suggest a potential upgrade from YOLOv3 (which the GitHub implementation uses) to YOLOv7 for 2D detection could improve the fit and, thus, the 3D estimations.

B. Traffic Lights:

In order to classify the arrows in the arrows in the traffic lights, we tried to color threshold the traffic lights and then get the corresponding contours, of the displayed shape (arrow). Due to different lighting conditions in different frames, the classical approach did not give good results.

We then implemented a working GitHub repo [7] which uses Yolo v3 trained on LISA Traffic Light Dataset for classifying arrows on the traffic lights.

C. Objects and Road Signs:

Additional objects like dustbin, traffic poles, traffic cones were detected using Detic and their respective 3D models were rendered in Blender.

The road signs which are the arrow heads on the road is detected by Mask-RCNN. Figure 5 illustrates the same.

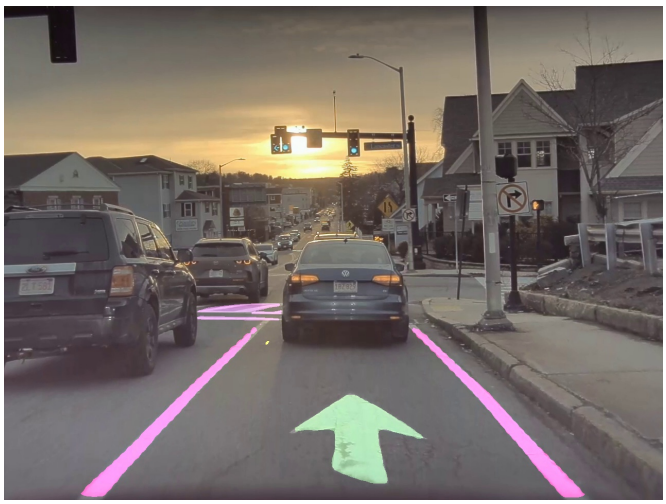

Fig. 5. Detection of road arrows using Mask RCNN

D. Pedestrian Pose:

For the task of identifying 3D pedestrian pose in the key frames, we implemented OSX [8] which gives us the 3D Mesh of the entire body. It processes both RGB images for human detection and corresponding depth maps to estimate the real-world position of each detected human. It uses a combination of off-the-shelf object detection (YOLOv5) and custom pose estimation to generate 3D meshes of humans in the scenes. Then 3D meshes are adjusted based on real-world coordinates derived from the depth maps, allowing for accurate placement in a 3D environment. Rendered images and .obj files for each detected human provide both a visual verification and a reusable 3D model for further applications. Figure 6 and 7 shows the detected 3D mesh of the person.

Fig. 6. 3D Mesh generation of pedestrian

Fig. 7. 3D Mesh generation of pedestrian

IV. Checkpoint III

A. Breaking Lights and Indicators of other vehicles

To detect brake lights, the following algorithm is employed, leveraging Detic's masking capabilities :

1) Extract braking lights from the image using masks and bounding box generated by Detic (although the confidence values of these boxes are really low -around 20%)
2) Find the 3D world points of these boxes using calibration matrix and depth map.
3) See which car's position are these 3D points (of the brake lights) using simple distance formulae.
4) Assign that car's braking state in blender as True.
5) Same goes for indicator lights

Figure 8 shows the detection of safety cones and braking lights

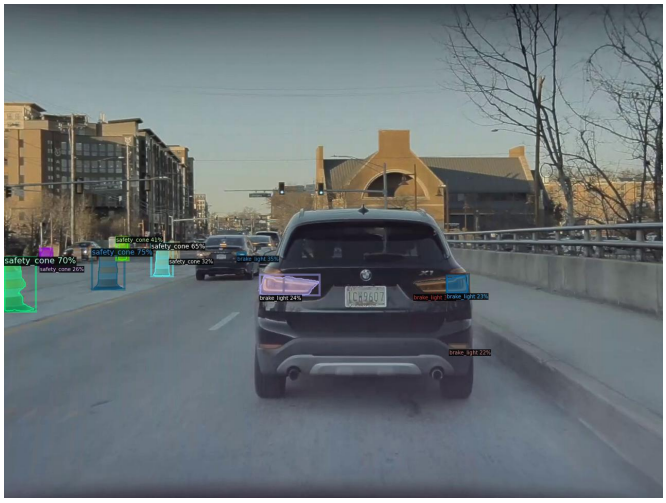

Fig. 8. Detection of Safety cones and braking light using Detic

B. Parked and Moving Vehicles:

To distinguish between parked and moving vehicles, we had to find optical flow of between consecutive key frames. Optical flow represents the motion of objects between two frames, often visualized as a field of arrows or a color-coded

image where the color and intensity indicate the direction and magnitude of motion, respectively. In order to that we implemented RAFT [9].

We get 2 channels (u, v) for for each pixel value, we calculate the magnitude and the directions of the resultant vector from its u, v giving. And based on threshold filtering, will decide which vehicles are moving and which are parked. Figure 9 and 10 illustrates the optical flow depth calculated using RAFT.

Fig. 9. Image (left) and Optical Flow(right)

Fig. 10. Image (left) and Optical Flow(right)

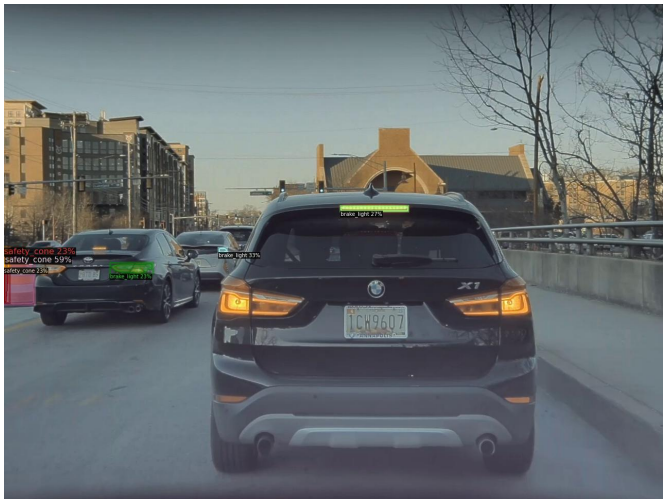

Fig. 11. Detection of Safety cones and braking light using Detic

REFERENCES

[1] Luigi Piccinelli, Yung-Hsu Yang, Christos Sakaridis, Mattia Segu, Siyuan Li, Luc Van Gool, and Fisher Yu. Unidepth: Universal monocular metric depth estimation, 2024.

[2] Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. Clrnet: Cross layer refinement network for lane detection, 2022.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[4] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Philipp Krähenbühl, and Ishan Misra. Detecting twenty-thousand classes using image-level supervision, 2022.

[5] S. Khadem. 3D Bounding Box Estimation Using Deep Learning and Geometry. https://github.com/skhadem/3D-BoundingBox. Accessed: 2023-09-30.

[6] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry, 2017.

[7] S. Khadem. 3D Bounding Box Estimation Using Deep Learning and Geometry. https://github.com/sovit-123/Traffic-Light-Detection-Using-YOLOv3. Accessed: 2023-09-30.

[8] Jing Lin, Ailing Zeng, Haoqian Wang, Lei Zhang, and Yu Li. One-stage 3d whole-body mesh recovery with component aware transformer, 2023.

[9] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow, 2020.

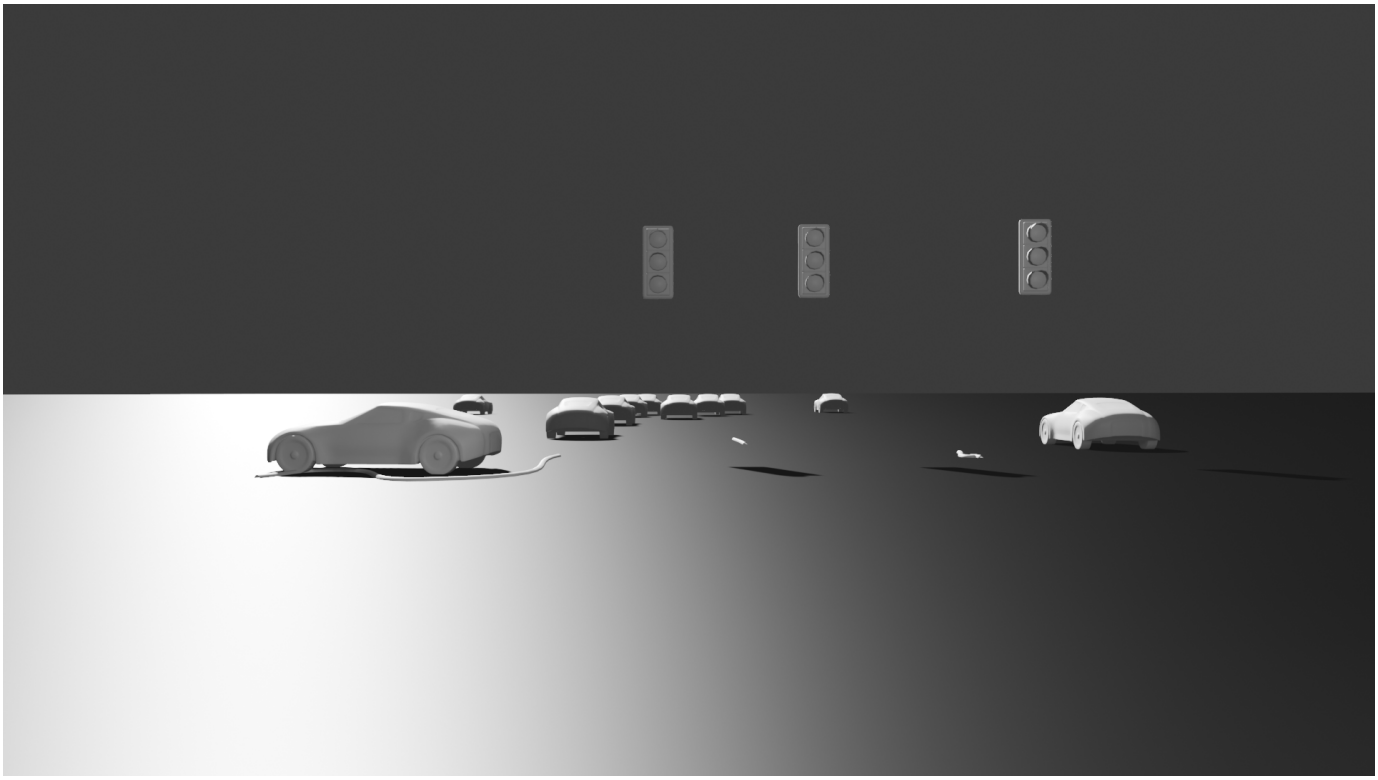

Fig. 12. Traffic Light Detection in Night Scene using Detic

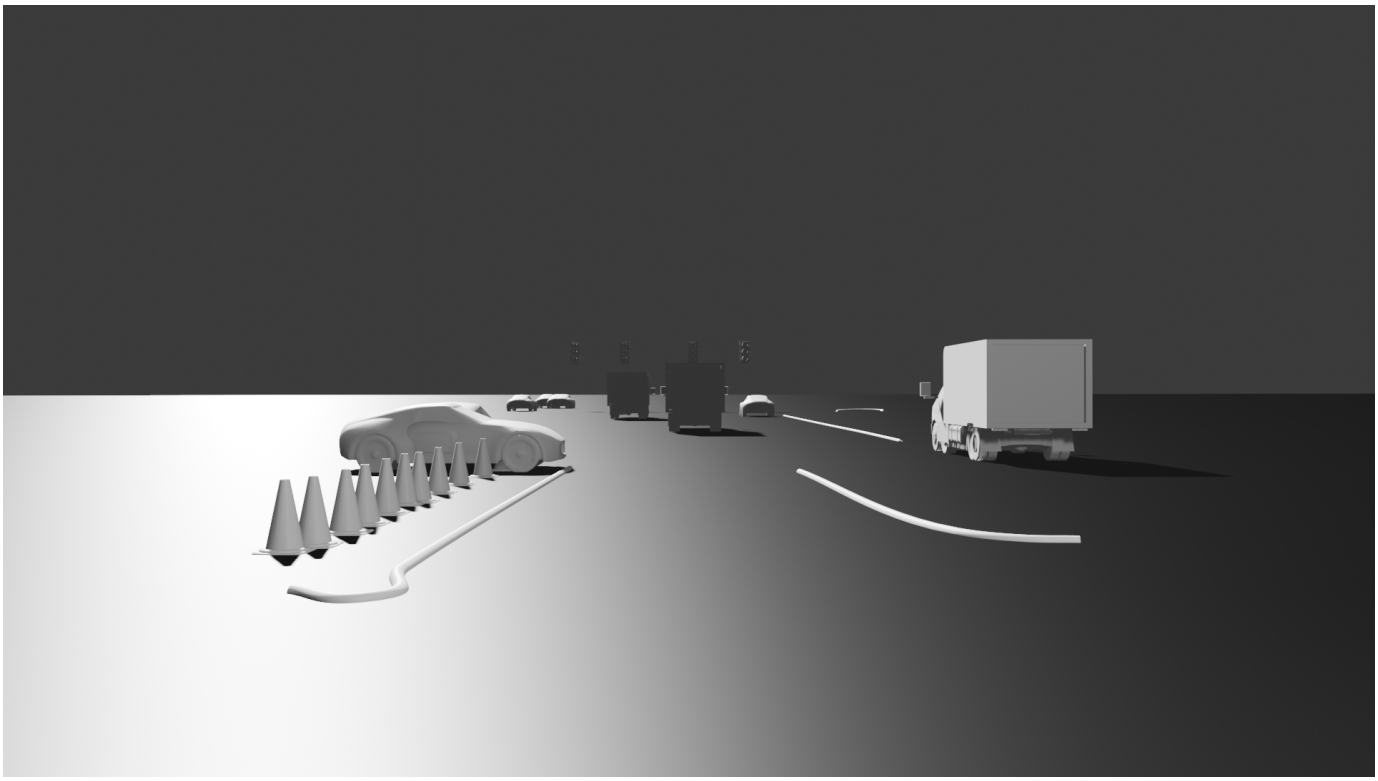

Fig. 13. Rendered image with cones and traffic lights

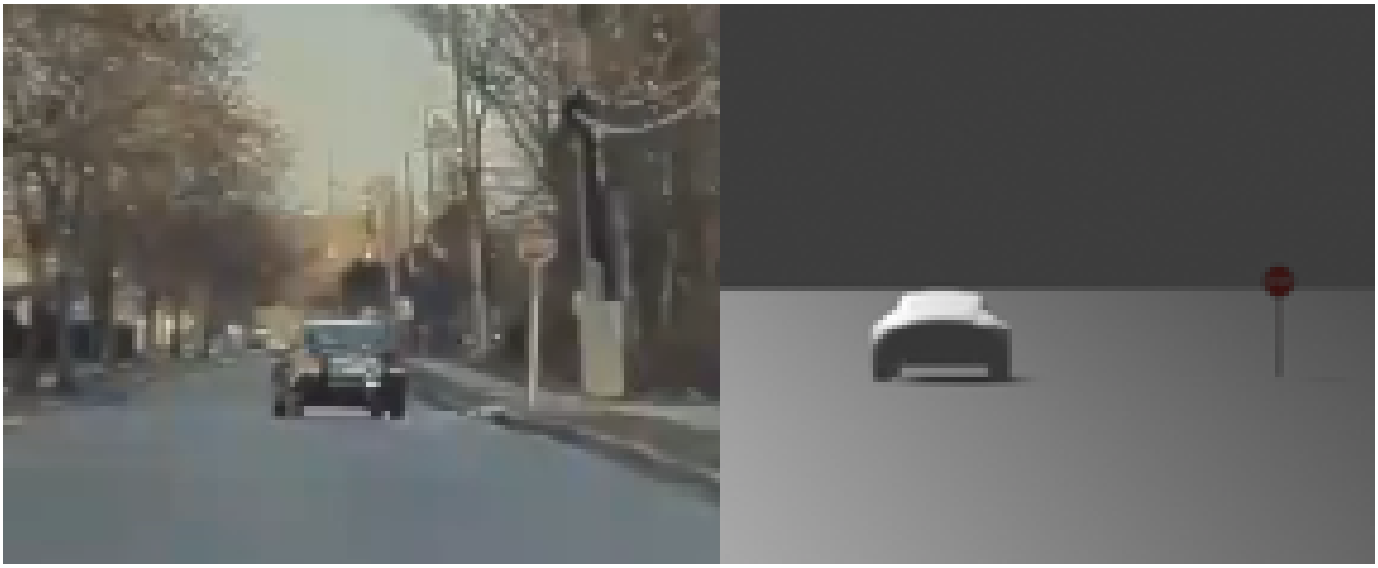

Fig. 14. Rendered image with stop light

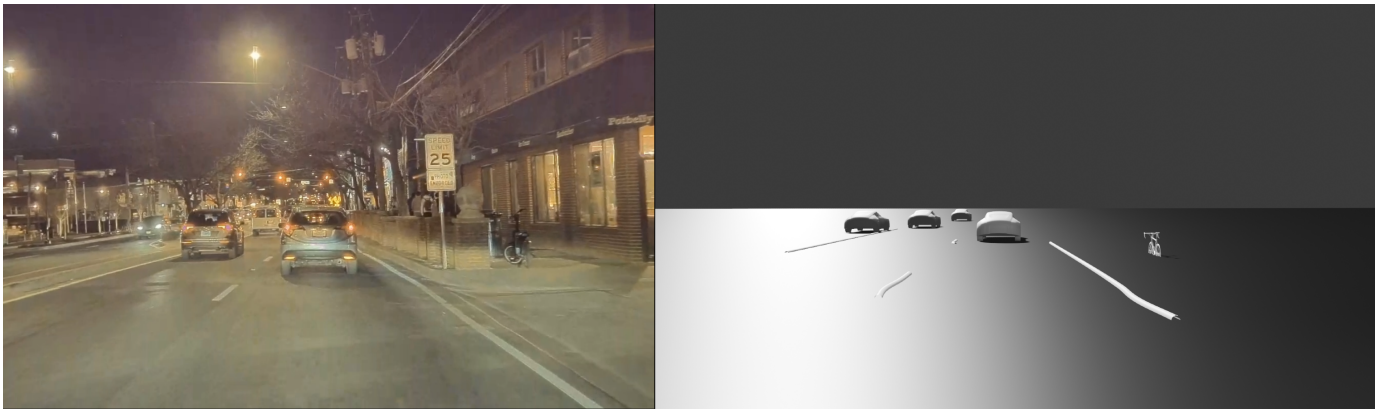

Fig. 15. Detection of parked cycle