EE267 – Lab 0: Access CARLA

Litian Gong, Hanhui Su

1 Environment Setup

Local machine snapshot:

Ubuntu 22.04
NVIDIA RTX 4070 super(12 GB VRAM)
CARLA version 0.9.15

2 Part 1: Running CARLA Server

The CARLA server was launched using Docker with GPU acceleration:

```
sudo docker run --privileged --gpus all --net=host -e DISPLAY=$DISPLAY
carlasim/carla:0.9.15
```

Once the server started, the simulator rendered the default city environment, confirming successful setup.

Figure 1: CARLA server successfully running

3 Part 2: Driving a Vehicle Using CARLA Client

Using the CARLA Python API, both manual and automatic control scripts were executed to interact with the simulator. Manual mode allows real-time keyboard control, while automatic mode enables autopilot functionality. Both use Pygame to visualize the car's sensor data and simulation state.

Figure 2: Manual control view

Figure 3: Automatic control using autopilot mode

4 Part 3: Using CARLA API: Traffic Manager

The Traffic Manager API was used to spawn and manage multiple vehicles and pedestrians in the environment. The traffic system can be run in synchronous mode to ensure deterministic results. By adjusting parameters like vehicle count, lane-changing behavior, and traffic light policies, diverse traffic scenes were generated.

Figure 4: Traffic Manager generating multiple vehicles

Figure 5: Traffic Manager generating multiple vehicles

5 Part 4: Bounding Boxes

This task involved projecting 3D bounding boxes from CARLA's world coordinates to the 2D image plane. The transformation and projection matrices were implemented using

NumPy and OpenCV. Bounding boxes were drawn around visible vehicles, demonstrating perception-level object localization.

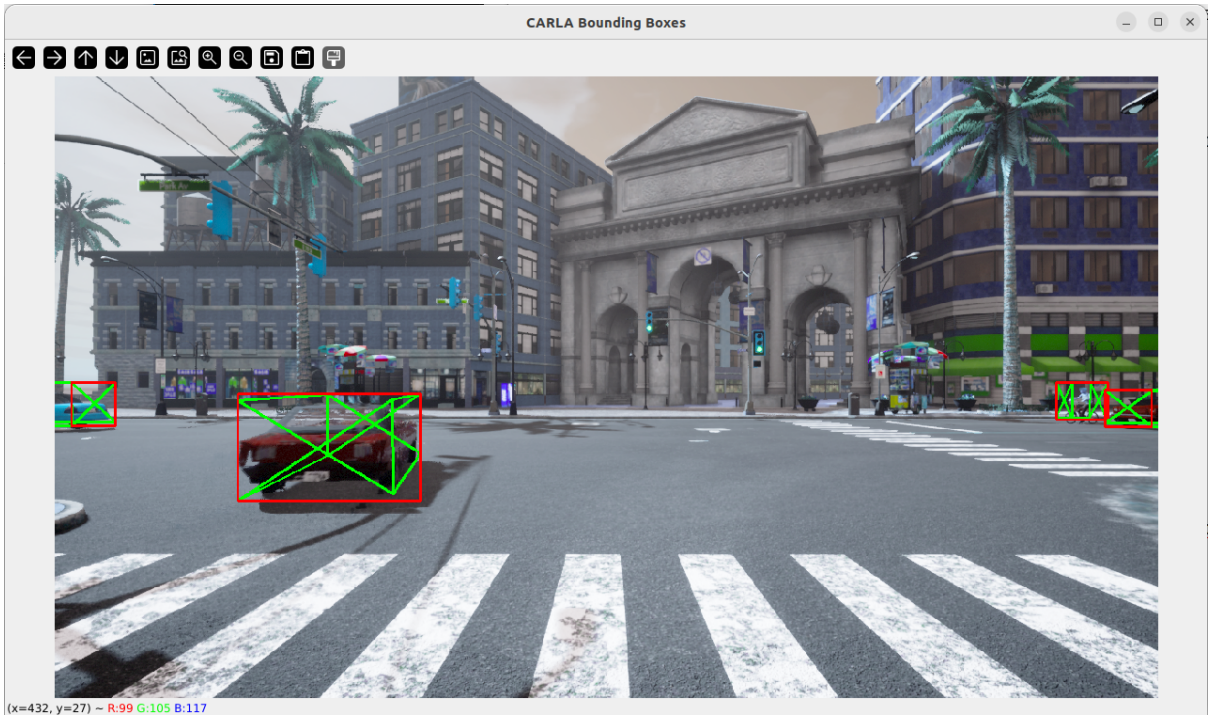

Figure 6: 3D bounding boxes projected to 2D image space

Figure 7: Multiple vehicles with bounding boxes shown in CARLA

6 Part 5: Instance Segmentation

In this part, an instance segmentation sensor was implemented using CARLA's Python API. The goal was to visualize unique object instances such as cars, pedestrians, and buildings, each with a distinct ID, providing richer perception data than standard semantic segmentation.

Figure 8: RGB scene image captured by the camera.

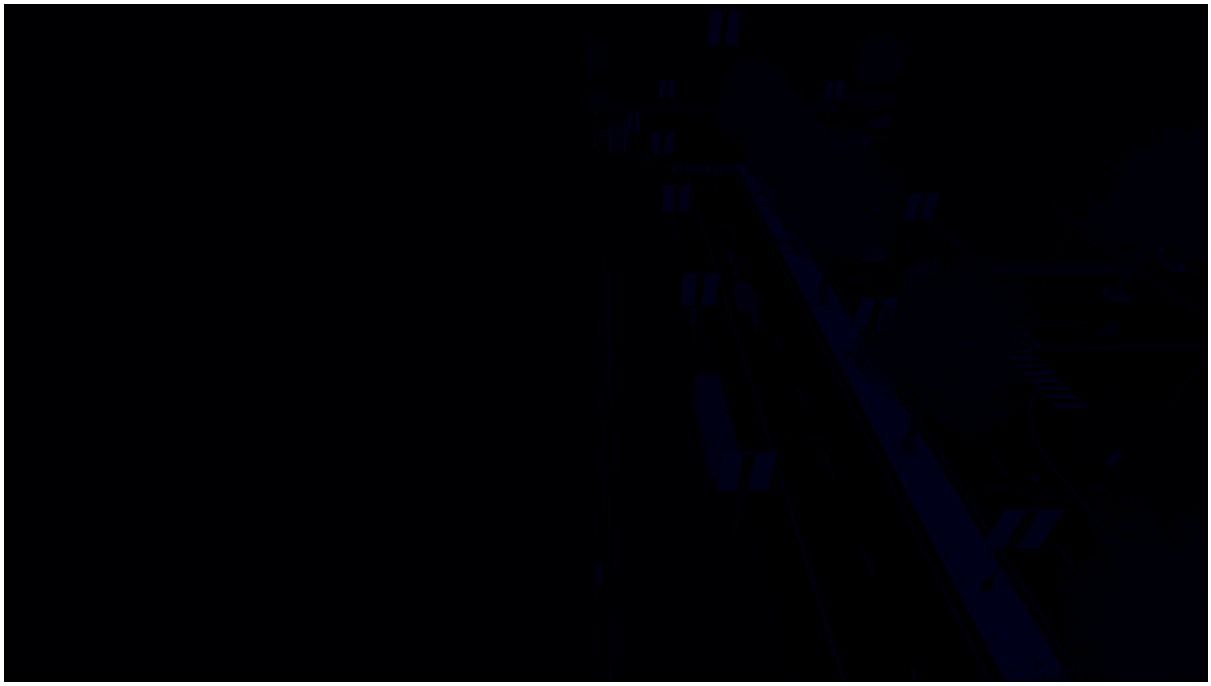

Figure 9: Semantic segmentation result. Each color corresponds to a class (e.g., vehicle, building, tree).

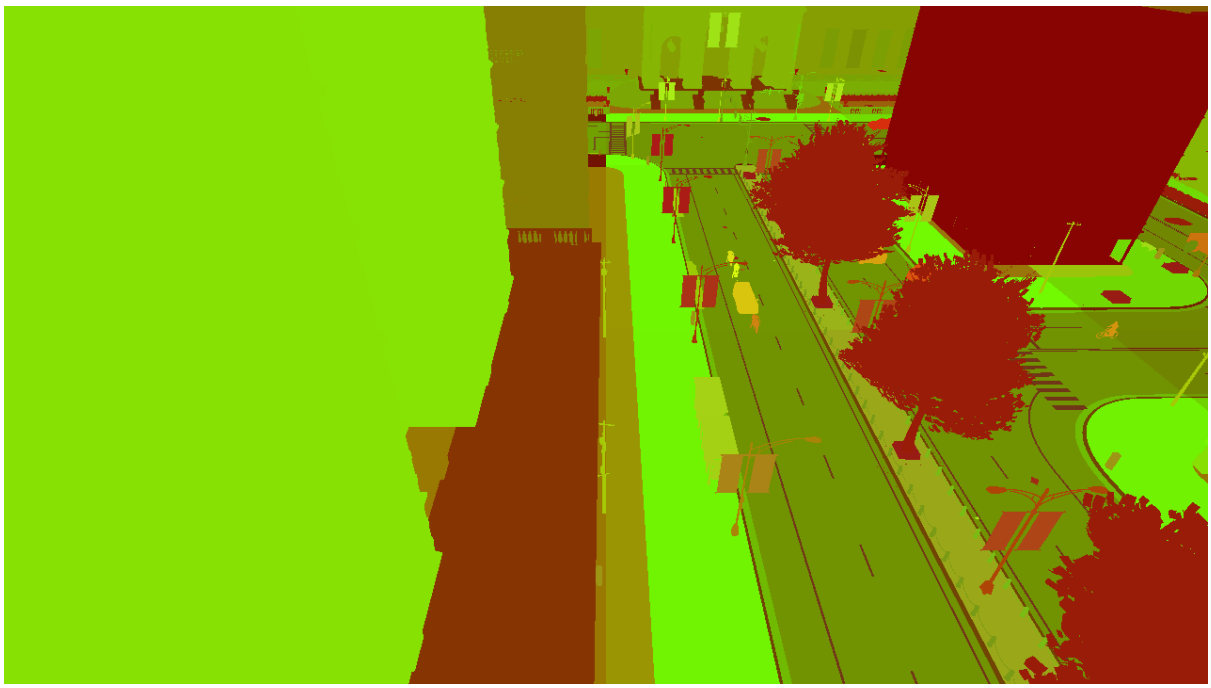

Figure 10: Instance segmentation result. Each color represents a unique object instance.

In the instance segmentation image, each object instance is encoded using a unique RGB triplet corresponding to its ID. Unlike semantic segmentation, which colors all cars the same, instance segmentation distinguishes each one individually.

Instance segmentation enables a more detailed understanding of the scene. While semantic segmentation provides general scene layout, instance segmentation provides

object-level granularity, which is critical for multi-object tracking, obstacle avoidance, and planning.