



浙江大学
ZHEJIANG UNIVERSITY

浙江大学软件学院优秀大学生夏令营
调研报告

姓 名 _____ 张宏远 _____

选 题 _____ 任务 1+子任务 1 _____

目录

一、 子任务 1.....	3
1.1 自动驾驶数据集调研报告	3
1.1.1 KITTI 数据集	3
1.1.1.1 KITTI 数据集调研	4
1.1.1.2 KITTI 数据集可视化	7
1.1.2 Waymo 数据集	10
1.1.2.1 Waymo 数据集调研	10
1.1.2.2 Waymo 数据集可视化	12
1.1.3 PandaSet 数据集	12
1.1.3.1 PandaSet 数据集调研	13
1.1.3.2 PandaSet 数据集可视化	13
1.1.4 Apollo 数据集	15
1.1.4.1 Apollo 数据集调研	15
1.1.5 SODA10M 数据集	21
1.1.5.1 SODA10M 数据集调研	21
1.2 自动驾驶数据集数据集比较	24
1.2.1 数据集特征对比分析	24
1.2.2 论文数据集对比	25
1.3 街景三维重建任务论述	27
1.3.1 数据处理	27
1.3.2 三维重现	28
1.3.3 模型优化	28
1.3.4 评估与验证	30
二、附录	31
三、引用	39

一、子任务 1

任务要求：在街景三维重建的任务中，通过自车传感器采集数据是至关重要的一环。当前，存在多个主流的自动驾驶数据集，如 KITTI、Waymo Open、PandaSet 和 nuScenes 等，它们为街景三维重建等计算机视觉任务提供了丰富的实际场景数据。请调研及比较上述例举的数据集，并针对街景三维重建任务进行论述。

完成概述：在 1.1.1 部分，我选取了五种数据集进行调研：KITTI、Waymo、PandaSet、Apollo 和 SODA10M。这些数据集涵盖了两个国外数据集、两个国内数据集以及一个中外合作数据集。对于每一个数据集我都从科研论文和网络信息两个角度进行了详细的调研，此外对于操作上可行的数据集，我还进行了数据集的可视化处理。在实际操作过程中，通过对数据集的深入理解和应用，提升了我对这些数据集的理解和掌握；在 1.1.2 部分，我对五种数据集进行了详尽的比较分析，这些数据集的比较基于科研论文和网络资料的详细调研。通过比较不同数据集的优缺点，可以更好地理解它们在自动驾驶研究和开发中的适用性和挑战；而在 1.1.3 部分，我详细阐述了街景三维重建任务的各个概念和步骤。这包括从激光扫描和摄影测量中获取数据，到利用计算机视觉和几何学算法进行点云配准和重建的过程。这一部分的目的是深入探讨街景三维重建在实际应用中的工作流程及其在自动驾驶和元宇宙领域的潜力应用。

1.1 自动驾驶数据集调研报告

在进行自动驾驶数据集的调研时，我选取了多个国内外的数据集，并对它们进行了详细的分析与应用。对于操作上可行的数据集，我不仅调研了相关背景和研究论文，还进行了实际应用测试。以下是对每个数据集的详细调研和应用介绍：

1.1.1 KITTI 数据集

KITTI 数据集已成为众多任务的标准训练和评估数据集，广泛用于 3D 物体检测、车道检测、立体重建、3D 分割、光流及融合方法（如 RGB 和点云数据融合）。此外，KITTI 还在评估 SLAM 和视觉里程计方法中得到广泛应用。数据集包括立体相机的 RGB 和灰度图像、IMU 数据和点云，所有模态的数据均同步采集，确保数据的一致性。KITTI 数据集采集了市区、乡村和高速公路等多种场景的真实图像数据，提供了多样化的环境，支持各种计算机视觉和机器学习任务的研究和开发。其下载地址为（<https://www.cvlibs.net/datasets/kitti/index.php>）。

1.1.1.1 KITTI 数据集调研

在 KITTI 数据集调研中,我参考了论文 *Vision meets Robotics: The KITTI Dataset* [1], 其论文链接为 (<https://www.cvlibs.net/publications/Geiger2013IJRR.pdf>)。

KITTI 数据集的记录平台如图 1 所示,这是一辆大众帕萨特旅行车配备了四个摄像头(两个彩色和两个灰度摄像头),一个旋转 3D 激光扫描仪和一个 GPS/IMU 组合惯性导航系统。在图 1 中就含有 KITTI 最重要是三个坐标系,蓝色坐标系是 velodyne 激光雷达坐标系,红色坐标系是相机坐标系,相机采集的图像是图像坐标系,在简要介绍完传感器后,我将对这三个坐标系进行详细的介绍。

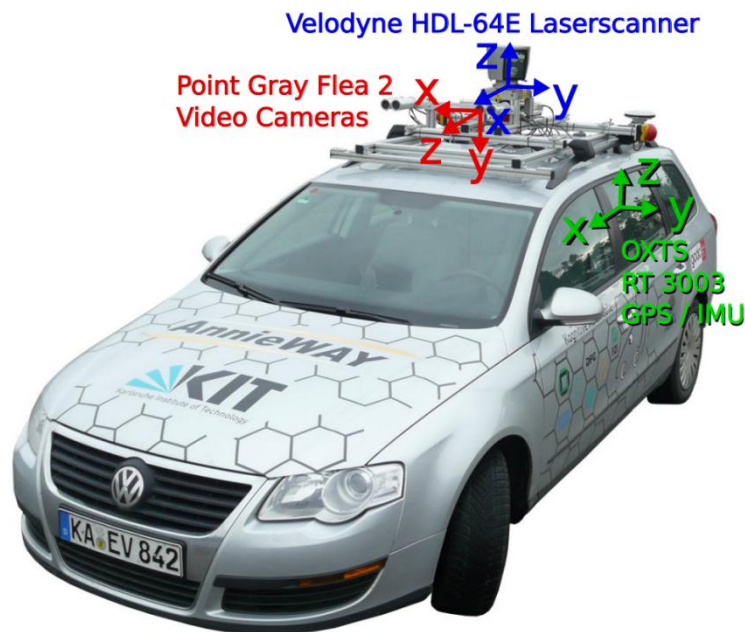


图 1 KITTI 数据集的记录平台

对于这样一个数据平台,其传感器设置如图 2 所示,其所包含的传感器如下列所示。

- 2 × PointGray Flea2 灰度相机 (FL2-14S3M-C), 1.4 百万像素, 1/2" Sony ICX267 CCD, 全局快门
- 2 × PointGray Flea2 彩色相机 (FL2-14S3C-C), 1.4 百万像素, 1/2" Sony ICX267 CCD, 全局快门
- 4 × Edmund Optics 镜头, 4mm, 开口角度约 90°, ROI 的垂直开口角度约 35°
- 1 × Velodyne HDL-64E 旋转 3D 激光扫描仪, 10 Hz, 64 束, 0.09 度角分辨率, 2 cm 距离精度, 收集约 130 万点/秒, 视场: 水平 360°, 垂直 26.8°, 范围: 120 m
- 1 × OXTS RT3003 惯性和 GPS 导航系统, 6 轴, 100 Hz, L1/L2 RTK, 分辨率: 0.02m / 0.1°

KITTI 采集中最重要的坐标系是相机坐标系和 velodyne 激光雷达坐标系。对

于相机坐标系，从图 2 可以清楚地看到，在相机坐标系中，z 轴指向光轴方向，即从镜头中心指向相机前方。x 轴指向右侧，y 轴指向下方，这一坐标系是右手坐标系。在激光雷达坐标系中，z 轴指向上方（朝向天空），x 轴指向扫描仪前方，y 轴指向左侧，这同样是一个右手坐标系。

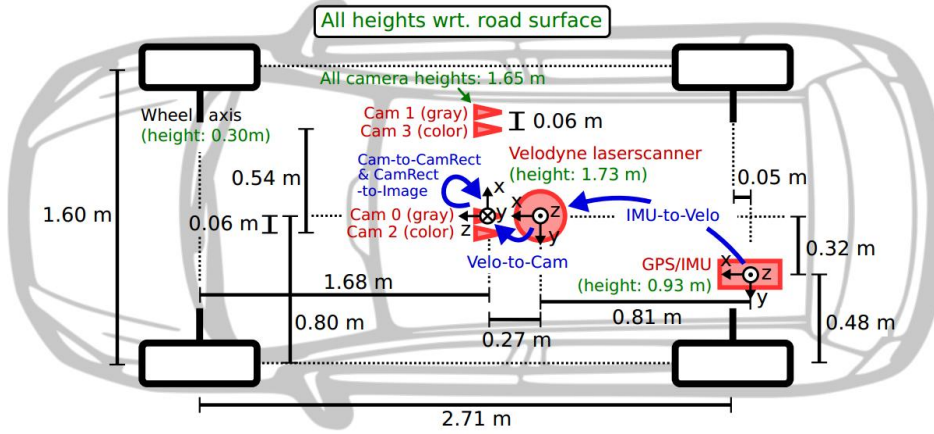


图 2 KITTI 数据集的传感器设置

在理解相机坐标系和 Velodyne 激光雷达坐标系后，可以开始介绍投影矩阵。在 KITTI 数据集中，每次通过同步传感器记录数据样本时，都会记录投影/变换矩阵。这些矩阵存储在 `calib/*.txt` 文件中，如图 3 所示。

```
P0: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 0.000000000000e+00
7.215377000000e+02 1.728540000000e+02 0.000000000000e+00 0.000000000000e+00
1.000000000000e+00 0.000000000000e+00 0.000000000000e+00 0.000000000000e+00

P1: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 -3.875744000000e+02
0.000000000000e+00 7.215377000000e+02 1.728540000000e+02 0.000000000000e+00
0.000000000000e+00 1.000000000000e+00 0.000000000000e+00 0.000000000000e+00

P2: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 4.485728000000e+01
7.215377000000e+02 1.728540000000e+02 2.163791000000e-01 0.000000000000e+00
1.000000000000e+00 2.745884000000e-03 0.000000000000e+00 0.000000000000e+00

P3: 7.215377000000e+02 0.000000000000e+00 6.095593000000e+02 -3.395242000000e+02
0.000000000000e+00 7.215377000000e+02 1.728540000000e+02 2.199360000000e+00
0.000000000000e+00 1.000000000000e+00 2.729905000000e-03 0.000000000000e+00

R0_rect: 9.999239000000e-01 9.837760000000e-03 -7.445048000000e-03 -9.869795000000e-03
9.999421000000e-01 -4.278459000000e-03 7.402527000000e-03 4.351614000000e-03
9.999631000000e-01 0.000000000000e+00 0.000000000000e+00 0.000000000000e+00

Tr_velo_to_cam: 7.533745000000e-03 -9.997140000000e-01 -6.166020000000e-04 -4.069766000000e-03
1.480249000000e-02 7.280733000000e-04 -9.998902000000e-01 -7.631618000000e-02
9.998621000000e-01 7.523790000000e-03 1.480755000000e-02 -2.717806000000e-01

Tr_imu_to_velo: 9.999976000000e-01 7.553071000000e-04 -2.035826000000e-03 -8.086759000000e-01
-7.854027000000e-04 9.998898000000e-01 -1.482298000000e-02 3.195559000000e-01
2.024406000000e-03 1.482454000000e-02 9.998881000000e-01 -7.987231000000e-01
```

图 3 `calib/*.txt`

在 KITTI 数据集中，P0、P1、P2 和 P3 是 3x4 的相机投影矩阵，分别对应于 P0=gray_L、P1=gray_R、P2=rgb_L 和 P3=rgb_R。R0_rect 是 3x3 的相机旋转矩阵，Tr_velo_to_cam 是 3x4 的激光雷达到相机的 RT 矩阵。

为了方便理解各坐标系之间的关系，我对这三种坐标系进行了手绘，具体关系如图 4 所示，并且具体的计算如公式 1 所示。

$$Z[u \ v \ 1]^T = P2 * R0_rect * Tr_velo_to_cam * [x \ y \ z \ 1]^T$$

公式 1

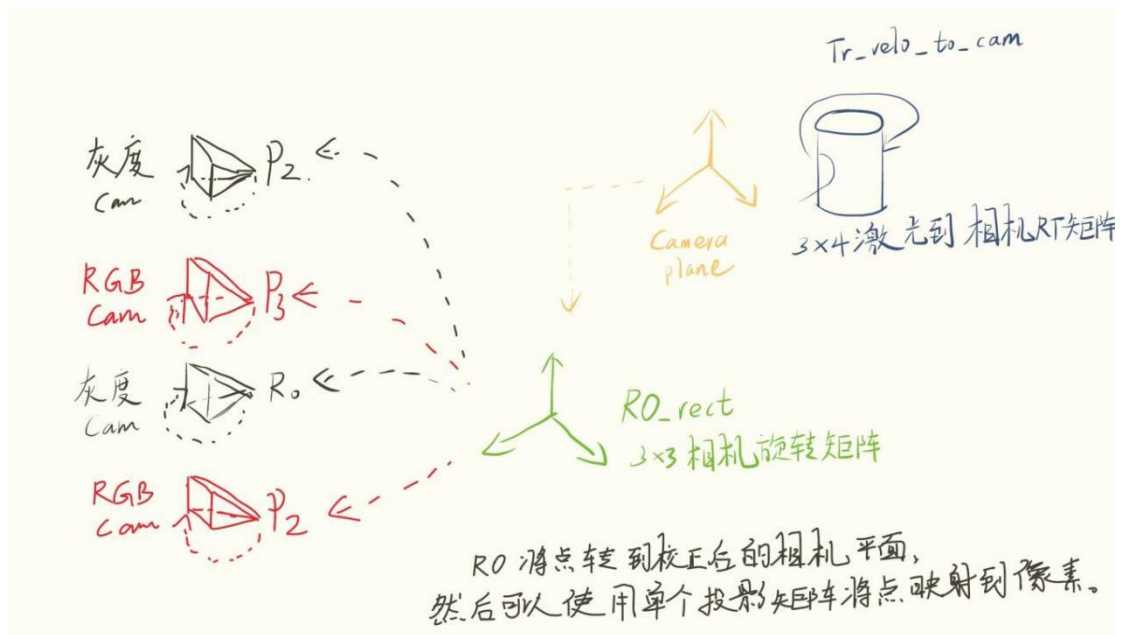


图 4 KITTI 中投影矩阵的结构粗略表示

介绍完基本的 KITTI 数据集中的概念，接下来我将对这个数据集进行应用展示。

1.1.1.2 KITTI 数据集可视化

论文中的 KITTI 数据集格式如图 5 所示，其中提供了 bin 二进制格式的点云数据。对于这些点云数据，首先需要解析二进制文件，将其转换为 numpy 格式，然后使用 open3d 进行可视化。也可以直接用 numpy 读取二进制文件。

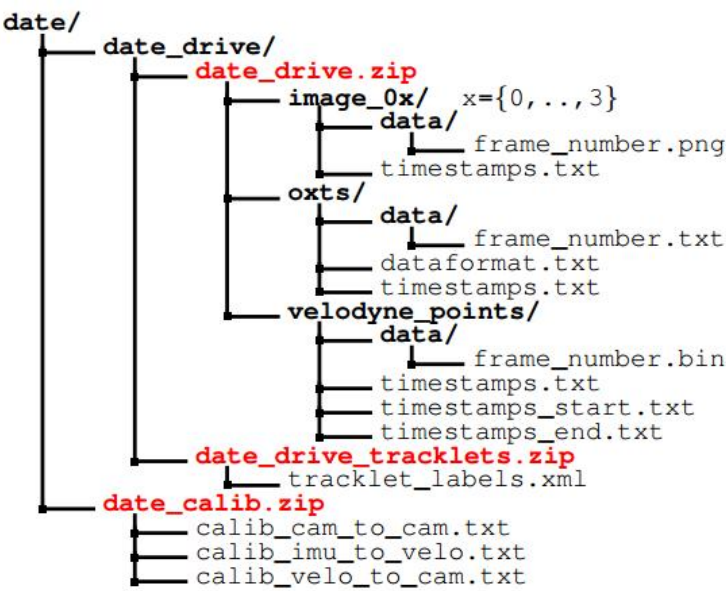


图 5 论文中给出 KITTI 数据集格式

在进行可视化步骤之前，需要在官网下载点云数据集，由于数据过大，这里我是在国内的转存资源库进行下载,同时需要在 anaconda 中配置 open3d 的编程环境，配置的部分环境如图 6 所示，项目工程数据集经过修改后格式后如图 7 所示。

```
nest-asyncio      1.6.0      pypi_0      pypi
numpy             2.0.0      pypi_0      pypi
open3d            0.18.0     pypi_0      pypi
openssl           3.0.14     h827c3e9_0
packaging         24.1       pypi_0      pypi
parso             0.8.4      pypi_0      pypi
pip               24.0       py310hba95532_0
```

图 6 Anaconda 配置环境

🔄 > 浙软 > python > KITTI_MAIN > 3D_py > data > kitti > training >

📄 📁 🗑️ 🔍 ⌵ 排序 🔍 查看 ⋮

名称	修改日期	类型	大小
calib	2024/7/6 10:13	文件夹	
image_2	2024/7/6 10:28	文件夹	
label_2	2024/7/6 10:26	文件夹	
velodyne	2024/7/6 10:05	文件夹	

图 7 可视化项目中 KITTI 数据集格式

KITTI 数据集中包含四类文件：**bin**（点云数据）、**rgb**（图像数据）、**label**（标注数据）和 **calib**（校准数据）。其中 **calib** 和 **label** 为 **txt** 文件，需要逐行读入并存入数组中，以便后续程序使用。对于 **kitti_Dataset.py** 的代码，其中包含了 **Object3d** 类、**Calib** 类、**Kitti_Dataset** 类（代码见附录代码 1）。

下面是展示点云的程序代码 **draw_3d.py** 对点云的渲染展示，（代码见附录代码 2）。



图 8 KITTI 点云展示样例 1-俯视图

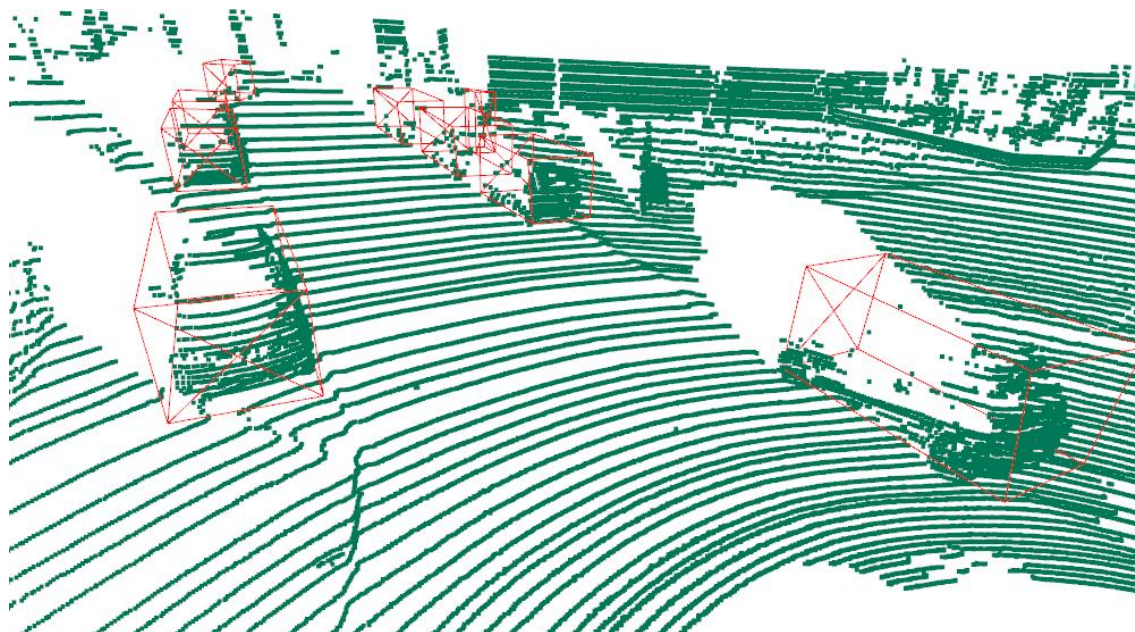


图 9 KITTI 点云展示样例 1-细节图

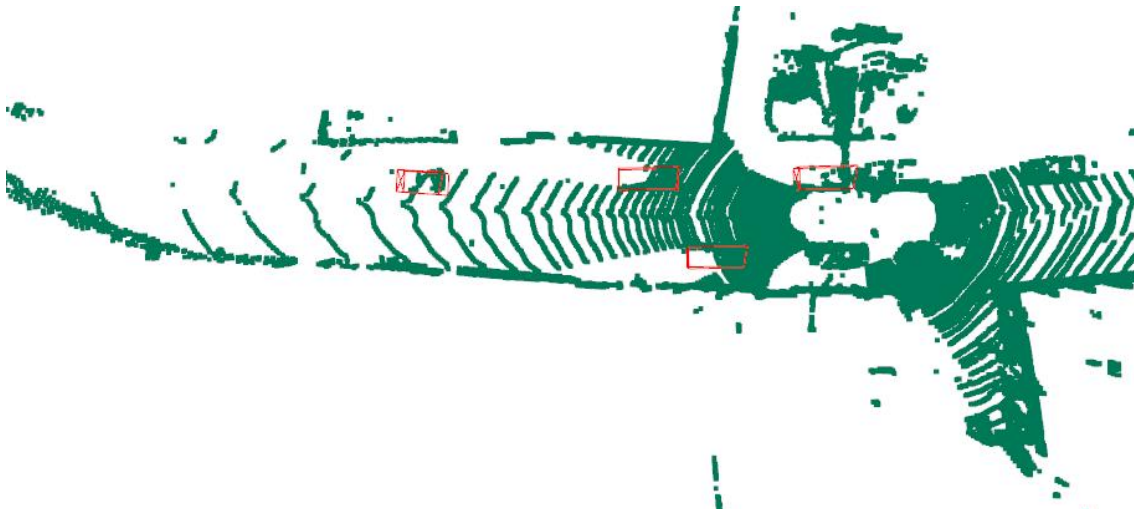


图 10 KITTI 点云展示样例 2-俯视图

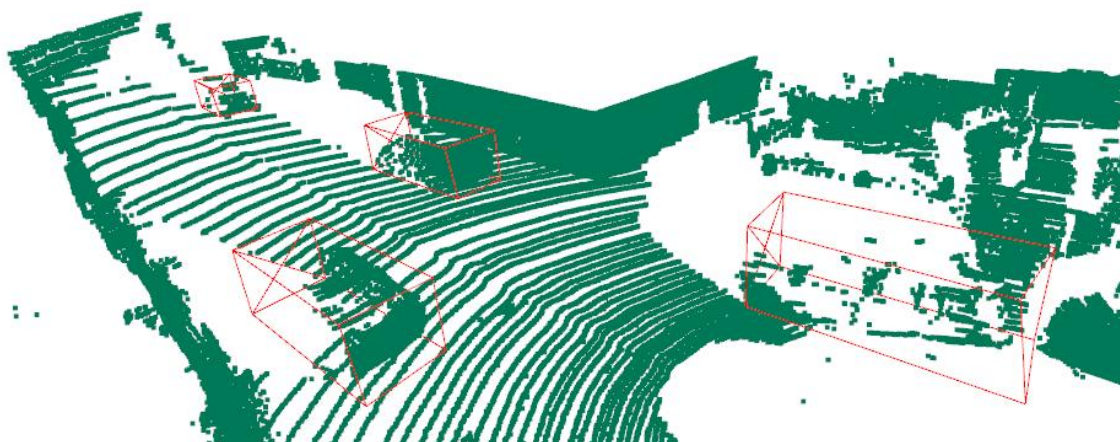


图 11 KITTI 点云展示样例 2-细节图

此外还使用 OpenCV 对 KITTI 数据集中的图像进行了 3D 框可视化（代码见附录代码 3 img_3dbox.py）。



图 12 KITTI 数据集 3D 框可视化

至此 KITTI 自动驾驶数据集部分展示完毕。

1.1.2 Waymo 数据集

2019 年 8 月 21 日，谷歌母公司 Alphabet 旗下的自动驾驶公司 Waymo 在其博客上宣布推出了 Waymo Open Dataset 项目。这个数据集包含 1150 个场景，每个场景时长为 20 秒，并且 LiDAR 和 Camera 数据经过同步和标定处理。图像和激光雷达的数据均经过详细的 bounding box 标注，并在各帧之间使用一致的标识符。

Waymo Dataset 分为两个部分：Perception Dataset 和 Motion Dataset。Perception Dataset 包含 2030 个场景，最新版本为 2022 年 6 月发布的 v1.4；Motion Dataset 包含目标轨迹追踪和 3D 同步地图，有 103354 个场景，最新版本为 2021 年 8 月发布的 v1.1。其数据集的下载地址为(<https://waymo.com/open>),另外国内浙江大学提供的的数据集下载地址为(<https://drive.google.com/file/d/1IJSFVGxwLLKLP8uidfUTGzb8n7nNDuli/view>)

1.1.2.1 Waymo 数据集调研

在 Waymo 数据集调研中，我主要参考了两篇科研论文，其中第一篇论文是 *Scalability in Perception for Autonomous Driving: Waymo Open Dataset* [2]，其论文链接为(<https://arxiv.org/abs/1912.04838v7>)，第二篇是 *Street Gaussians: Modeling Dynamic Urban Scenes with Gaussian Splatting* [3]，其论文链接为(<https://arxiv.org/abs/2401.01339>)。

与 KITTI 数据集采集中的传感器不同，Waymo 数据集数据收集过程利用了五个 LiDAR 和五个高分辨率针孔相机完成。LiDAR 数据受到了范围限制，并且每个激光脉冲都包含双回波信息。相机图像通过卷帘快门扫描捕获，具体的扫描模式可能因场景而异。所有相机图像都经过降采样，并从原始图像中裁剪而来，以适应特定的应用需求和处理流程。图 13 展示了相机水平视场（HFOV），它描述了相机传感器在其 x-y 平面上的视角范围，通常以 x 轴角度的形式表示。这个角度范围指定了相机可以从左到右扫描或捕捉的视野范围，这对于理解相机在特定场景中能够覆盖的区域至关重要。HFOV 的大小直接影响了相机的视觉覆盖范围和应用在场景中的适用性，因此在设计和部署视觉系统时，这个参数通常被仔细考虑和调整。

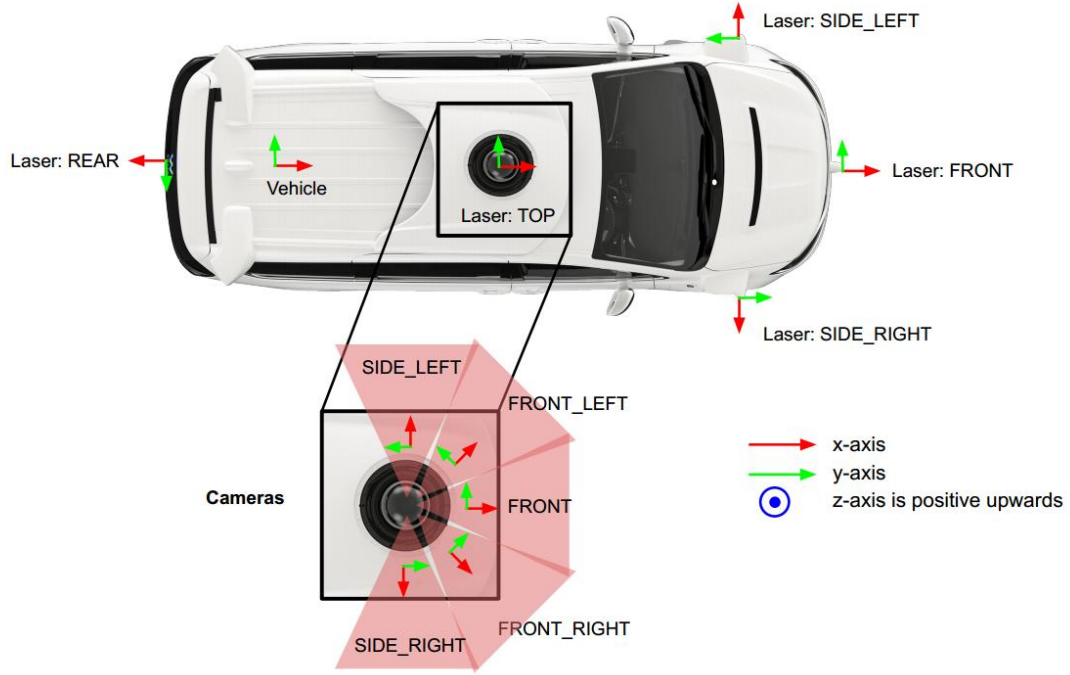


图 13 Waymo 数据集传感器布局和坐标系

整个数据集中采用右手法则来定义坐标系。全局坐标系使用 ENU 坐标系：Up(z)轴与重力方向一致，向上为正；East(x)沿着纬度指向正东，North(y)指向北极。车辆坐标系随着汽车的移动而变化，其中 x 轴指向车辆的前方，y 轴指向车辆的左侧，z 轴向上为正。传感器坐标系可以通过旋转矩阵从车辆坐标系获得，这些矩阵通常称为外参矩阵。

激光雷达使用球面坐标系，基于其在激光雷达传感器框架中的笛卡尔坐标系。相反，图像坐标是二维的，其中 x 轴表示图像的宽度，y 轴表示图像的高度，坐标原点位于图像的左上角。激光雷达测得的点的笛卡尔坐标可以表示为(x, y, z)，通过公式 2 可以唯一地转换为激光雷达球面坐标系中的距离、方位和倾角的元组。

$$\begin{aligned}
 range &= \sqrt{x^2 + y^2 + z^2} \\
 azimuth &= \text{atan2}(y, x) \\
 inclination &= \text{atan2}\left(z, \sqrt{x^2 + y^2}\right)
 \end{aligned}$$

公式 2

在该数据集中，对汽车、行人、交通标志和自行车等物体进行了详细的标注。针对激光雷达数据，每个物体被标注为一个 7 自由度的 3D 边界框 (bbox)，其中包括中心坐标 (cx, cy, cz)、长度 (l)、宽度 (w)、高度 (h) 和偏航角 (θ)。此外，每个物体还被分配了一个唯一的追踪 ID 编号。如图 14 所示。

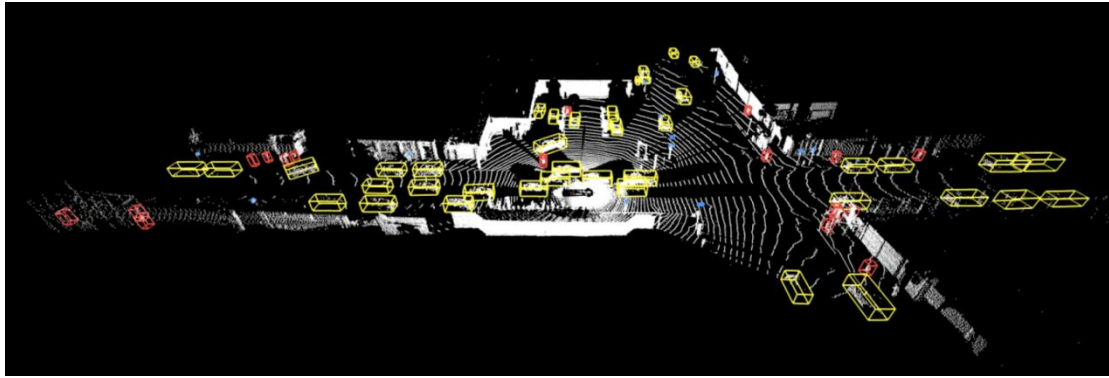


图 14 Waymo 激光雷达标签示例。黄色=车辆、红色=行人、蓝色=符号、粉红色=自行车手

在图像标注中，每个物体被标注为一个 4 自由度的 2D 边界框，包括中心图像坐标 (cx, cy)、长度 (l) 和宽度 (w)。

此外，数据集将标注的物体分为两个难度级别：LEVEL_2 表示物体在激光雷达数据中对应的点数少于 5 个，而其余的物体则划分为 LEVEL_1，这种分级有助于进一步的数据分析和算法评估。

在 Waymo 数据集中，LiDAR 数据以距离图像的形式进行编码，每个 LiDAR 返回一张距离图像，而每张图像包含了前两次返回波的数据。距离图像的格式类似于卷帘快门相机图像，按照从左到右逐列填充的方式排列。每个像素点对应 LiDAR 传感器的一个返回波。图像的高度和宽度由 LiDAR 传感器框架中倾角和方位角的分辨率决定，这种编码方式有效地将 LiDAR 数据转换为图像形式，便于后续的视觉处理和分析。

Waymo 数据集任务主要包括 2D 和 3D 物体检测以及追踪任务。训练集包含 798 个场景，用于训练和优化算法模型，验证集包含 202 个场景，用于调整模型超参数和评估模型的泛化能力，而测试集则包含 150 个场景，用于最终评估和比较不同算法在真实场景中的表现。

1.1.2.2 Waymo 数据集可视化

由于我选择了 Waymo 数据集作为子任务 3 的重点研究对象，我会在子任务 3 详细探讨 Waymo 自动驾驶数据集的可视化部分。因此，在这个章节中，我不会再重复详细描述该数据集的可视化方法和技术。

1.1.3 PandaSet 数据集

禾赛科技与人工智能数据标注平台公司 Scale AI 联合发布了 PandaSet 数据集，这是一款面向 L5 级自动驾驶的开源商用数据集。该数据集不仅适用于训练机器学习模型，还在推动自动驾驶技术的实现方面发挥着重要作用。PandaSet

数据集的显著特点之一是首次同时采用了机械旋转和图像级前向两种激光雷达进行数据采集。这种多角度采集方法不仅为数据提供了丰富的视角，还能够生成高质量的点云分割结果，有助于进一步提升自动驾驶系统的感知能力和准确性。

数据集的发布旨在服务科研和商业应用领域，为研究人员和开发者提供了一个重要的数据资源，用于探索新的自动驾驶技术和算法，并支持商业应用中的验证和实施。通过 PandaSet 数据集，用户能够获取到广泛的场景数据和丰富的标注信息，这对于构建更加健壮和可靠的自动驾驶系统具有重要意义。数据集下载地址为(<https://scale.com/open-av-datasets/pandaset>)。

目前访问 PandaSet 数据集官方网站会显示 “No matching content found” 的信息。据查阅资料，自 2021 年以来，该数据集无法从官方网站上获取。然而，通过其他渠道，可以找到该数据集的备份或相关资源。例如，数据集还可以在 Kaggle 等平台上找到，可以提供额外的获取途径和下载选项，Kaggle 地址为 (<https://www.kaggle.com/datasets/usharengaraju/pandaset-dataset/data>)

1.1.3.1 PandaSet 数据集调研

PandaSet 数据集的主要目标是支持机器学习模型的训练，推动自动驾驶技术的发展和实现。该数据集首次采用了机械旋转和图像级前向两类激光雷达进行数据采集，这种多角度采集方法能够生成精确的点云分割结果，适用于科研和商业应用领域。数据集包含超过 48,000 个摄像头图像和 16,000 个激光雷达扫描点云图像，涵盖了超过 100 个 8 秒钟的场景。每个场景都附带 28 个注释以及大多数场景的 37 个语义分割标签，这些详细的标注信息为数据分析和模型训练提供了重要支持。

在数据采集过程中，使用的车辆是克莱斯勒，配备了多种传感器，包括 1 个机械 LiDAR、1 个固态 LiDAR、5 个广角摄像头、1 个长焦摄像头，以及板载的 GPS 和 IMU 系统。这些高质量的数据和丰富的传感器配置使 PandaSet 数据集成为研究新技术和验证自动驾驶系统的理想选择。

1.1.3.2 PandaSet 数据集可视化

由于 PandaSet 数据集的官方网站不再提供数据下载，并且相应的数据集加载工具包 `pandaset-devkit` 在 4 年前已经停止了维护，我决定选择了官方网站提供的 8 个场景来进行 PandaSet 数据集的可视化。这些场景可能是通过官方网站提供的预览或演示数据来展示数据集的特征和内容。

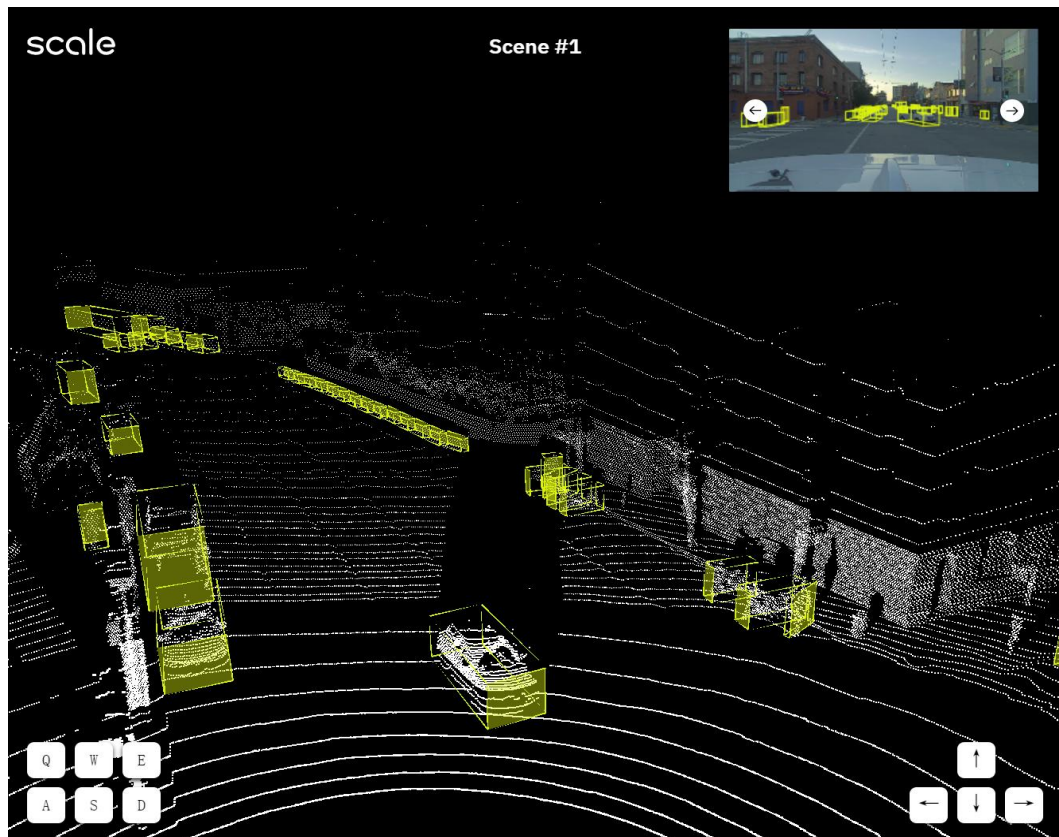


图 15 PandaSet 数据集场景 1

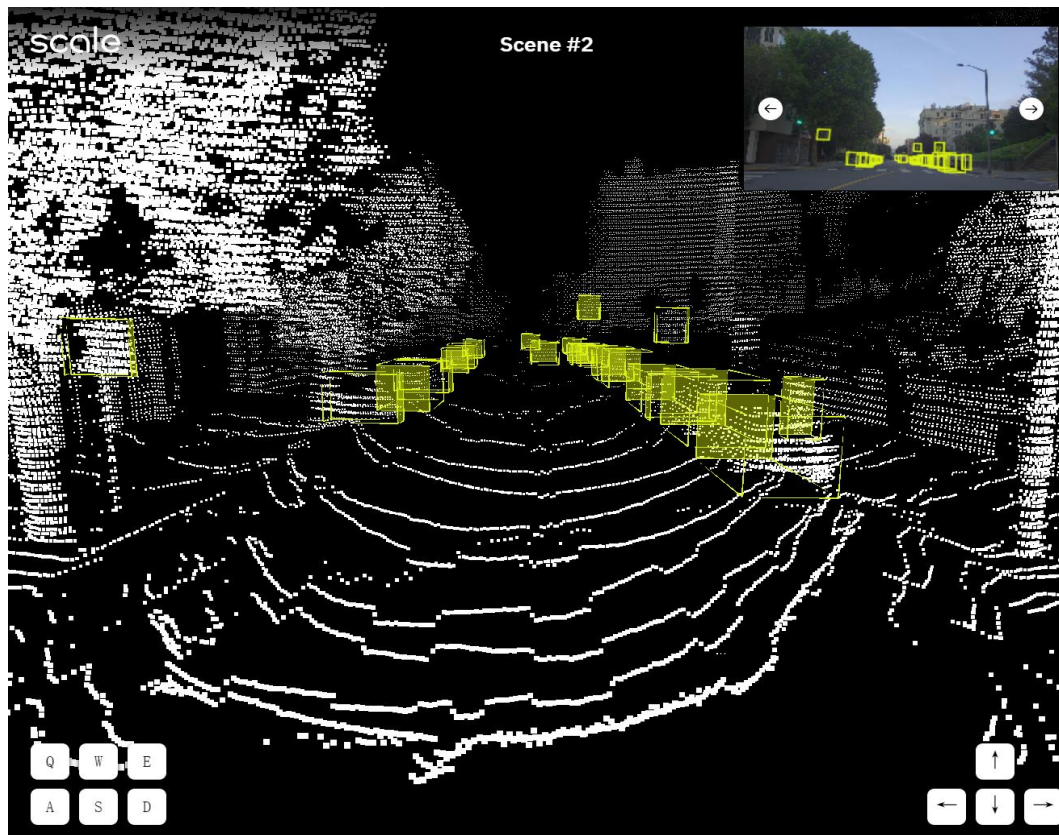


图 16 PandaSet 数据集场景 2

1.1.4 Apollo 数据集

Apollo（阿波罗）是百度推出的开放、完整、安全的自动驾驶平台，旨在帮助汽车行业和自动驾驶领域的合作伙伴快速构建自己的自动驾驶系统。作为开放平台，Apollo 强调开放能力、共享资源、加速创新和持续共赢的理念。

百度将其强大、成熟、安全的自动驾驶技术和丰富的数据资源开放给整个行业，旨在建立一个以合作为核心的生态系统。这一举措旨在利用百度在人工智能领域的技术优势，为合作伙伴提供支持，共同推动自动驾驶产业的发展和 innovation。官方提供的 Apollo 下载地址可通过访问官网获取，其具体地址如下：

(https://developer.apollo.auto/southbay_cn.html)，其数据集数量太过庞大，总计将近占用 800G 的存储空间，由于电脑配置限制，所以本章主要是对 Apollo 数据集进行理解调研，不涉及应用部分。

1.1.4.1 Apollo 数据集调研

Apollo 是百度 Apollo 自动驾驶开放平台的专题项目之一，是目前行业内环境最复杂、标注最精准、数据量最大的三维自动驾驶数据集。在调研的过程中我主要参考了三篇论文，第一篇 *The ApolloScape Open Dataset for Autonomous Driving and its Application* [4], 论文链接为(<https://arxiv.org/abs/1803.06184>)。第二篇论文是 *TrafficPredict: Trajectory Prediction for Heterogeneous Traffic-Agents* [5], 论文链接为(<https://arxiv.org/abs/1811.02146>)。最后是 *DVI: Depth Guided Video Inpainting for Autonomous Driving* [6], 论文链接(<https://arxiv.org/abs/2007.08854>)。

Apollo 与类似的数据集如 Cityscapes 和 KITTI 相比，ApolloScape 的数据量超过 10 倍，涵盖了感知、仿真场景和路网数据等多个方面，包括数十万帧逐像素语义分割标注的高分辨率图像数据、稠密点云、立体图像和立体全景图像。这些数据集成了更复杂的环境、天气条件和交通状况，为自动驾驶算法的开发和验证提供了丰富的场景和数据资源。Apollo 不仅开放了大量数据，还涵盖了逐帧标注的视频序列和三维背景，提供了中国最复杂的街景，每张图片内可以包含上百个车辆和行人。这些数据的开放旨在为全球的研究者、开发者和企业提供一个重要的研究和实验平台，促进自动驾驶技术的进步和创新。在 Apollo 数据集采集的过程中，会采用 Riegl VMX-1HA 作为主要设备获取静态的 3D 环境数据，如图 17 所示。该系统包括两个 VMX-1HA 激光扫描仪（具有 360°视场，测量范围为 1.2 米至 420 米，目标反射率大于 80%）、一个 VMX-CS6 相机系统（配备两个分辨率为 3384×2710 的前置摄像头）以及一个集成了 IMU/GNSS 的测量头（位置精度在 20 至 50 毫米之间，滚转和俯仰精度为 0.005°，航向精度为 0.015°）

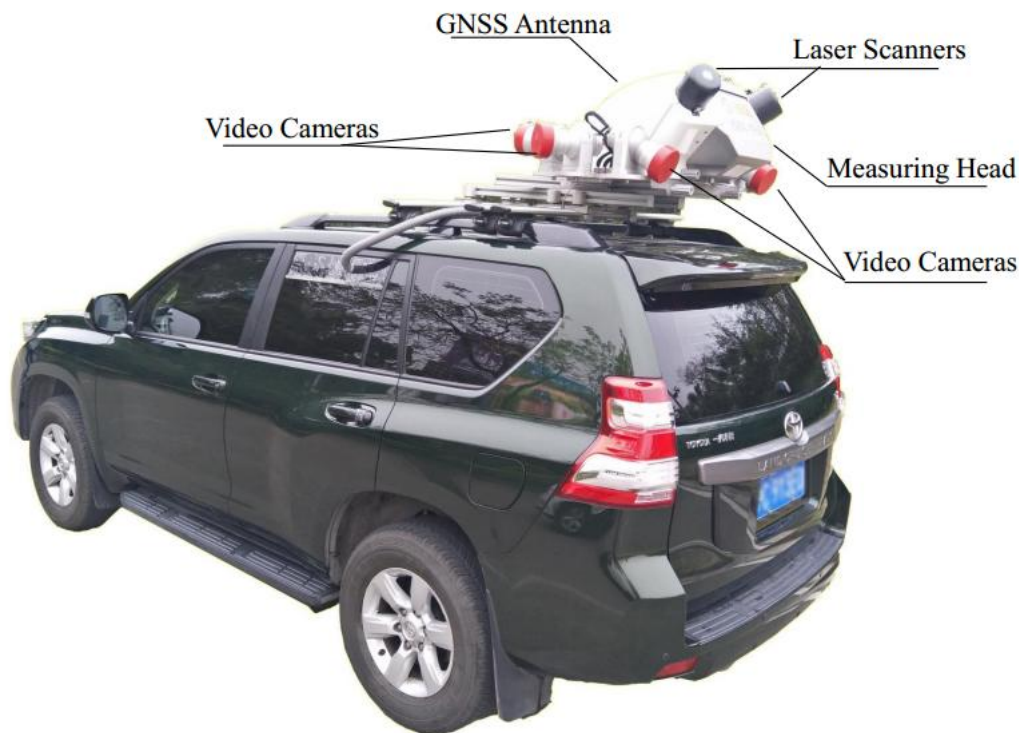


图 17 Apollo 数据集采集系统

激光扫描仪利用两束激光垂直扫描周围环境，类似于推扫帚相机的工作原理。与常用的 Velodyne HDL-64E 相比，Riegl VMX-1HA 扫描仪能够获取更高密度的点云数据，从而提升环境数据的精确度和细节呈现（5 毫米/3 毫米）。整个系统已进行内部校准和同步，并安装在中型 SUV 的顶部。

Apollo 数据集内容丰富，涵盖了八个主要任务：场景解析（Scene Parsing）、3D 车辆实例分割（3D Car Instance）、车道线分割（Lane Segmentation）、自定位（Self Localization）、轨迹预测（Trajectory）、3D 激光雷达物体检测和跟踪（3D Lidar Object Detection and Tracking）、立体估计（Stereo Estimation）以及图像修复（Inpainting）。接下来我将对这八个任务进行简要介绍。

场景解析（Scene Parsing）：如图 18 所示，任务的目标是理解输入图像中的场景，并对每个像素进行分类，将其分配到特定的类别中。对于自动驾驶而言，场景解析有助于识别和理解道路上的各种元素，如道路、行人、车辆和建筑等，这对于智能车辆进行高级决策和规划至关重要。场景解析旨在为图像中的每个像素或点云中的每个点分配一个类别（语义）标签，是对 2D/3D 场景最全面的分析之一。随着自动驾驶技术的发展，环境感知被视为关键的技术支持。百度公司提供的 ApolloScope 数据集包括高分辨率图像和逐像素标注的 RGB 视频、具有语义分割的测量级密集 3D 点云、立体视频和全景图像。数据集使用一辆配备高分辨率摄像头和 Riegl 采集系统的中型 SUV 在不同城市和各种交通条件下采集，移

动物体（如车辆和行人）的数量从几十个到一百多个不等。此外，每张图像都带有厘米级精度的高精度位置信息，静态背景点云的相对精度达毫米级。

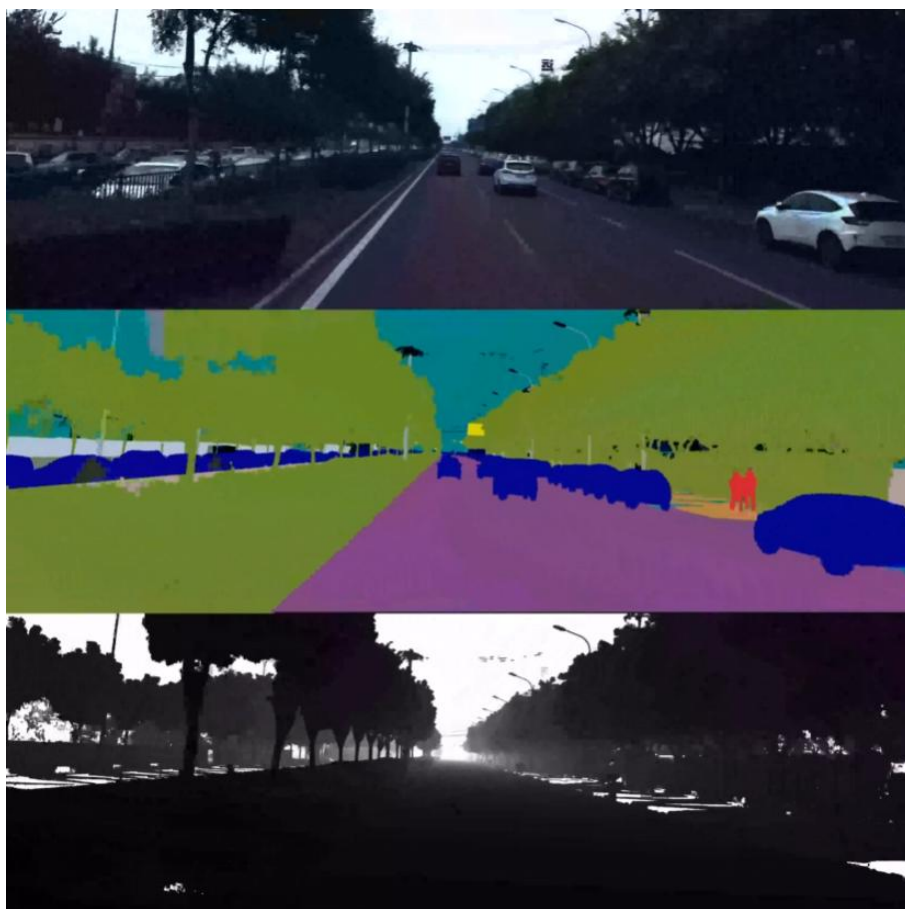


图 18 Apollo 数据集场景解析

3D 车辆实例分割（3D Car Instance）：这项任务旨在对行车记录图像中的车辆进行像素级别的分割，以准确区分不同车辆的边界并将其高亮显示。对于自动驾驶汽车而言，关键是能够有效检测到其他车辆、行人、骑车人等各种物体。系统必须能够深入理解每个图像帧中每个物体的三维关系，尤其是那些环绕或接近自动驾驶车辆的物体。



图 19 Apollo 数据集 3D 车辆实例分割

车道线分割（Lane Segmentation）：如图 20 所示，精确的高清地图（HD 地图）通常包含车道标线等要素，是商用自动驾驶车辆导航系统的后端基础。目前，大多数 HD 地图都是由人工标注员手动构建的。这些分割的结果可以直接用于 HD 地图的构建或更新过程中，从而帮助自动驾驶车辆准确地定位、导航和路径规划，确保车辆能够安全稳定地行驶在道路上。

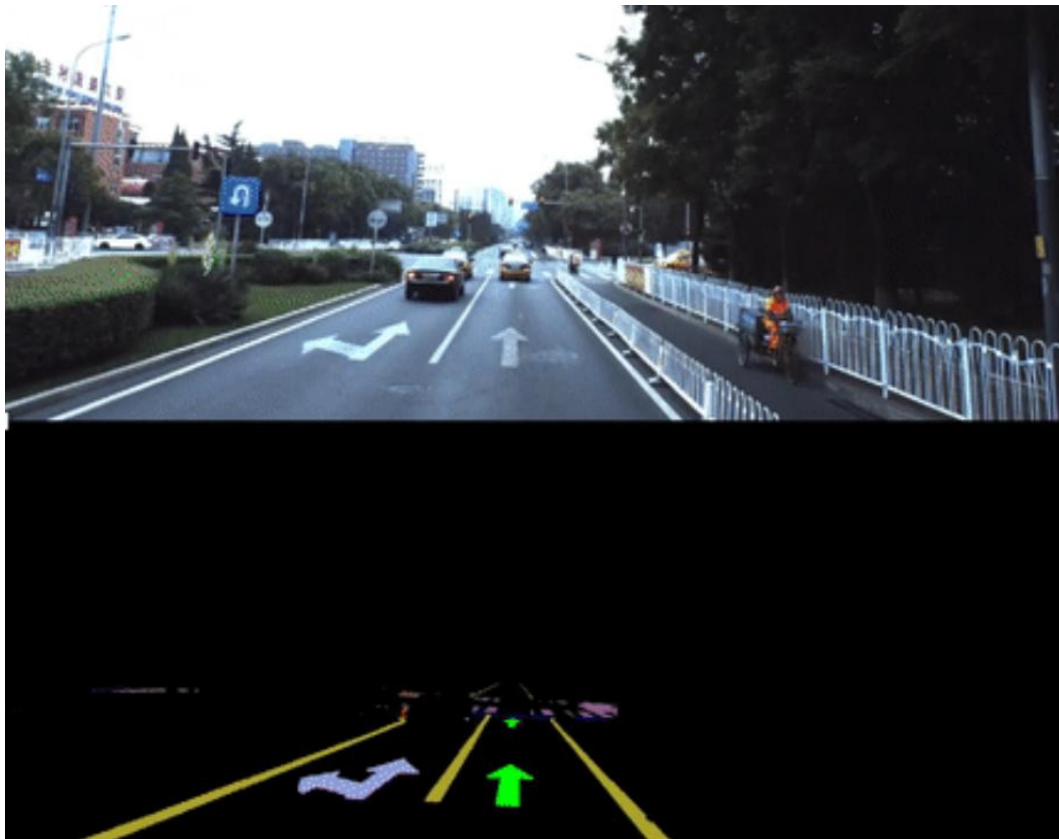


图 20 Apollo 数据集车道线分割

自定位（Self Localization）：如图 21 所示，任务的主要目标是确定车辆在世界坐标系中的精确位置。这是通过传感器信息获取周围环境场景，并实现车辆自我定位来实现的。在自动驾驶领域，准确的自我定位对于实现精确导航和路径规划至关重要，帮助车辆准确理解自身在道路网络中的位置。

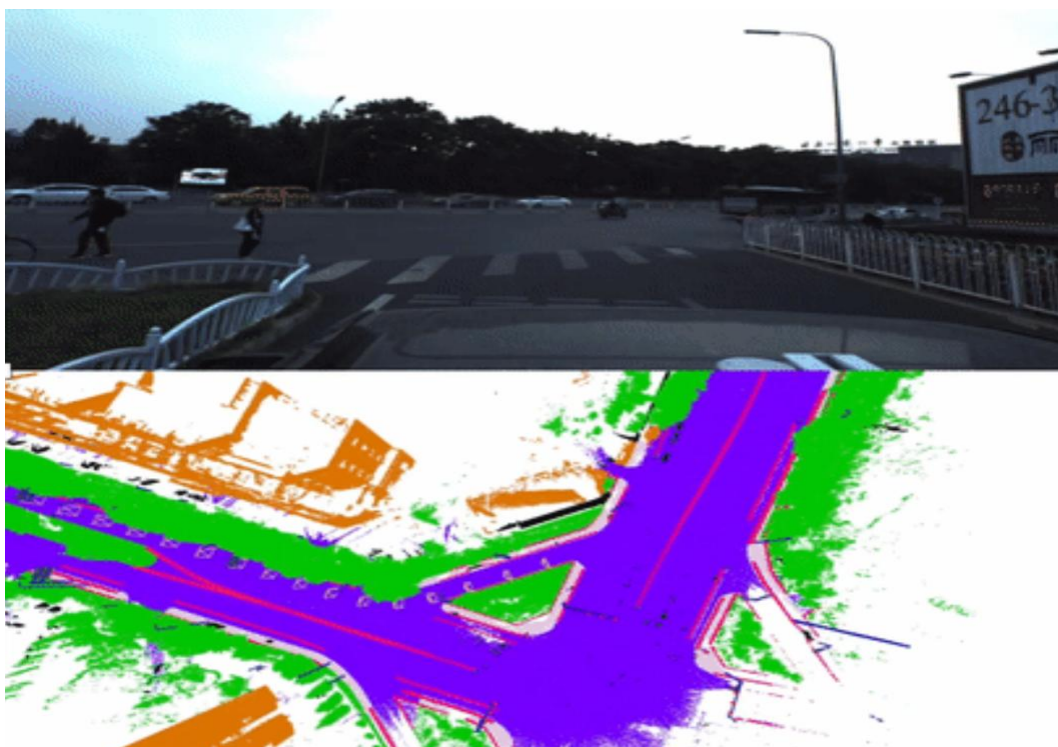


图 21 Apollo 数据集自定位

轨迹预测 (Trajectory)：如图 22 所示，轨迹任务涉及对运动物体（如车辆、行人等）可能采取的路径进行建模和预测。通过提前预测其他交通参与者的行为，可以更好地规划车辆的行驶轨迹。轨迹数据集包含了传感器识别的周围物体的坐标点集及其详细信息，如长宽高等，然后通过人工标注的方式对这些点集进行分类（如车辆、行人等）。这些数据集在多种光照条件和交通密度下进行收集，涵盖了高度复杂的交通流，包括混合了车辆、骑车人和行人的场景。

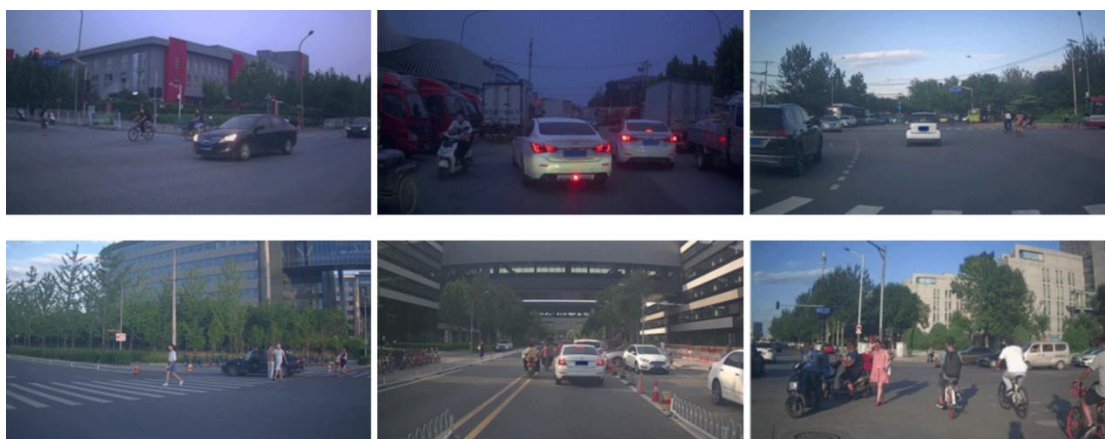


图 22 Apollo 数据集轨迹预测

3D 激光雷达物体检测和跟踪 (3D Lidar Object Detection and Tracking)：如图 23 所示，任务涉及检测和跟踪两个主要方面。检测任务旨在识别车辆周围存在的各种对象，如车辆、行人和交通标志等。跟踪任务则专注于在时间序列中对这些对象的运动进行跟踪和预测。数据集本质上是基于激光扫描的点云数据，通过

精细的人工标注确定了各个检测对象的位置、形状和大小。数据集是在北京市不同光照条件和交通密度下采集的高质量激光扫描点云数据，并进行了详细的标注。具体而言，数据集包含了高度复杂的交通流，涉及到各种车辆、骑车人和行人。

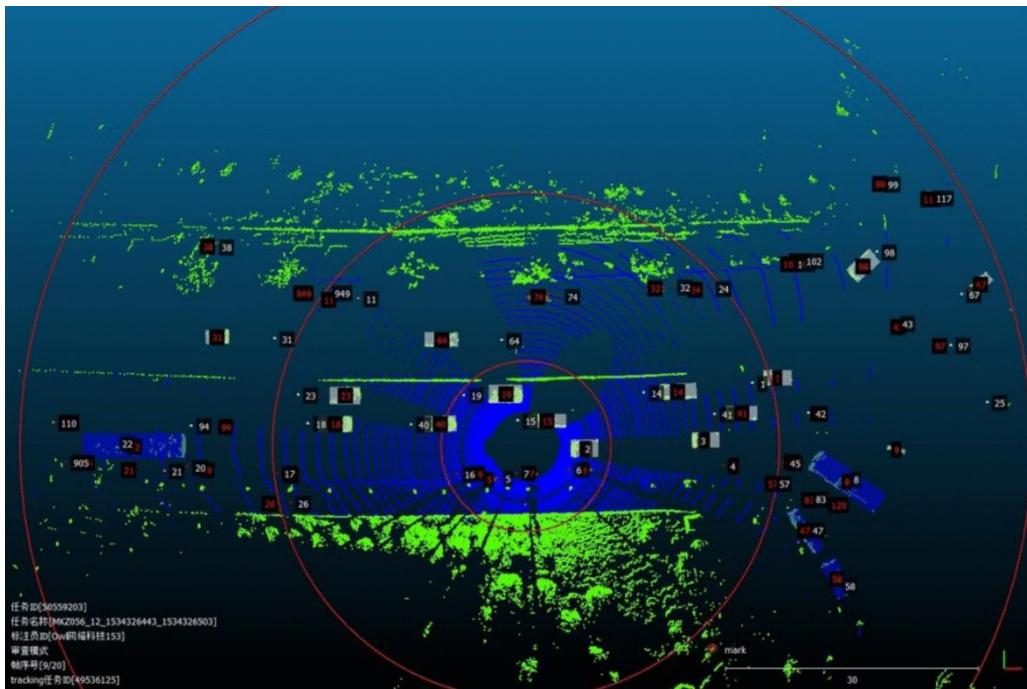


图 23 Apollo 数据集 3D 激光雷达物体检测和跟踪

立体估计（Stereo Estimation）：如图 24 所示，这部分任务专注于从立体图像对中推断深度信息，即场景中不同点到相机的距离。数据集包含 5165 对图像及其对应的视差图，其中 4156 对用于训练，1009 对用于测试。这些图像对来自 Apollo 数据集。为了获取地面真实数据，研究人员通过累积来自激光雷达的 3D 点云，并将 3D CAD 模型拟合到各自移动的汽车上（这些信息源自 3D 车辆实例理解数据集）。

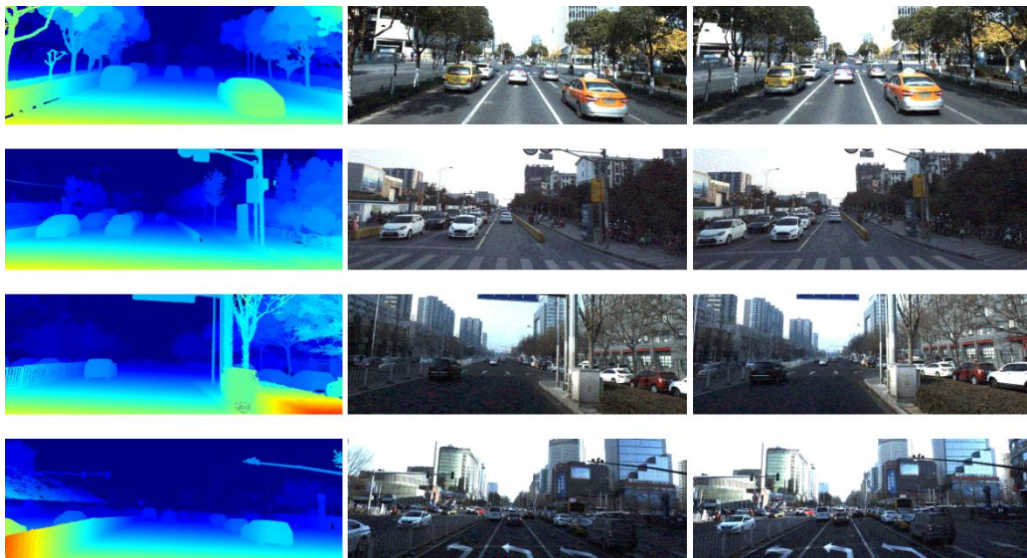


图 24 Apollo 数据集立体估计

图像修复（Inpainting）：如图 25 所示，在自动驾驶领域中，场景解析和修复任务是至关重要的。在实际道路环境中，由于多种原因（例如天气条件、遮挡或传感器故障），可能会导致场景不完整或缺失。因此，需要对这些缺失或不完整的场景进行修复，以确保自动驾驶系统能够准确地感知和理解周围环境。场景解析旨在从感知数据中准确识别道路、车辆、行人等重要元素，而修复任务则专注于填补或纠正这些信息的不足，从而支持自动驾驶系统在复杂场景下做出正确的决策。

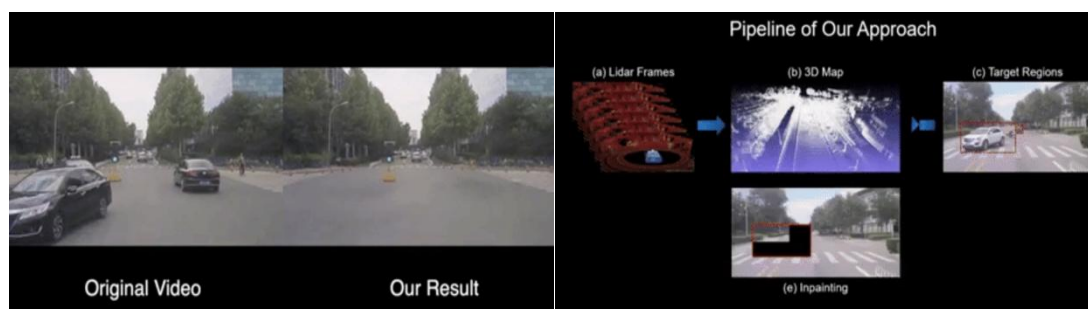


图 25 Apollo 数据图像修复

1.1.5 SODA10M 数据集

SODA10M 数据集是华为诺亚方舟实验室联合中山大学发布了新一代半/自监督的 2D 基准数据集，主要包含从 32 个城市采集的一千万张多样性丰富的无标签道路场景图片以及两万张带标签图片。其目的是补全自动驾驶领域仍缺乏一个基于大量无标签数据与少量有标签数据构成的可用于同时评价不同半/自监督方法的基准数据集的缺口。其下载地址为(<https://soda-2d.github.io/download.html>)。

1.1.5.1 SODA10M 数据集调研

由于 SODA10M 是一个较新的数据集，网络上其相关信息较少，所以本部分调研资料主要来源于这篇关于 SODA10M 的科研论文 *SODA10M: A Large-Scale 2D Self/Semi-Supervised Object Detection Dataset for Autonomous Driving* [7]，论文链接为(<https://arxiv.org/abs/2106.11118>)。

SODA10M 是一款高质量的行车场景数据集，通过众包分发模式采集，并且所有相关的隐私信息，如车牌号和人脸，都已经按照相关标准进行了模糊化处理。在众多数据中不乏有各种挑战性的环境下，包括 1000 万张覆盖不同天气条件、时间段和地点的图像。如图前三列图像来自 SODA10M 的标记集，最后一列图像来自未标记集。这图 26 所示，些图像提供了丰富的多样性，有助于提升自动驾驶模型在不同场景下的泛化性能。

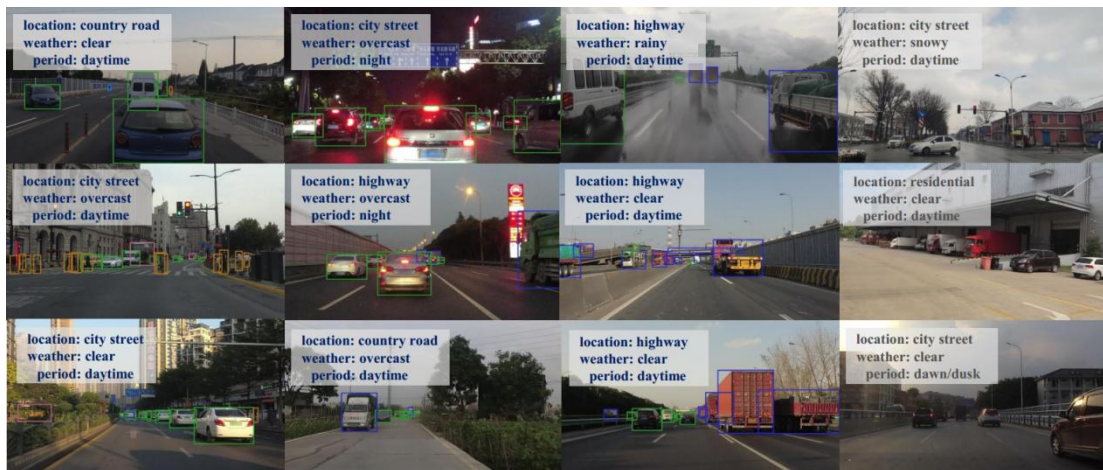


图 26 有挑战性的道路环境

这个数据集主要分为两个部分：一千万张无标签图片和两万张有标签图片。无标签图片的分布特性和数据属性已经在图一中进行了详细的分析。有标签图片则涵盖了 6 种主要的人车场景类别，包括行人、骑车人、汽车、卡车、有轨电车和三轮车。千万级无标签图片的数据特性分析如图 26 所示。

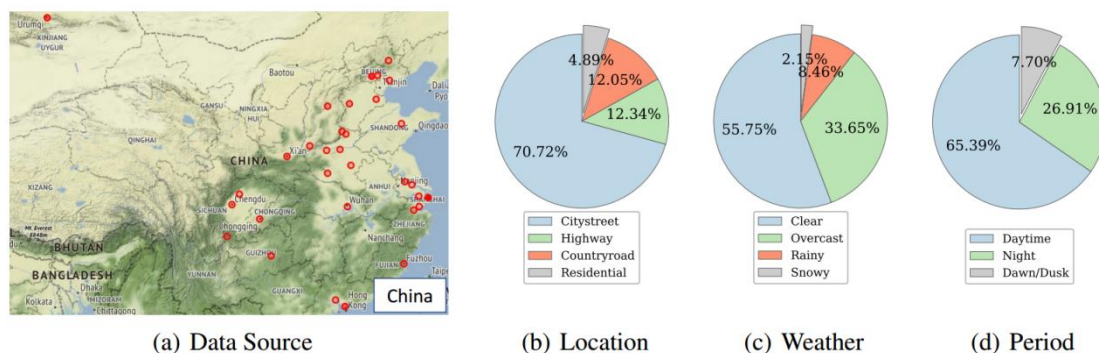


图 27 无标签数据集统计信息(a)数据来源的地理分布(b)每个地点的图像数目(c)每种天气情况下的图像数目(d)每个时期的图像数量。

图 26 显示，为了保持数据的丰富性和多样性，无标签图片从 32 个不同城市中选取，覆盖了中国的大部分区域。这些图片包含多种道路场景，如城市道路、高速公路、城乡道路和园区，以及不同天气条件，包括晴天、多云、雨天和雪天。同时，图片还涵盖了不同的时间段，包括白天、晚上和凌晨/黄昏。如此丰富的多样性确保了数据集在自监督预训练和半监督学习的额外数据上具有良好的泛化性能，适用于各种下游自动驾驶任务。

该数据集还提供了两万张带标注的图片，用于快速验证相关算法的性能。其数据统计特征如图 28 所示。丰富的标注数据使得研究人员能够有效地评估和优化各种自动驾驶算法。

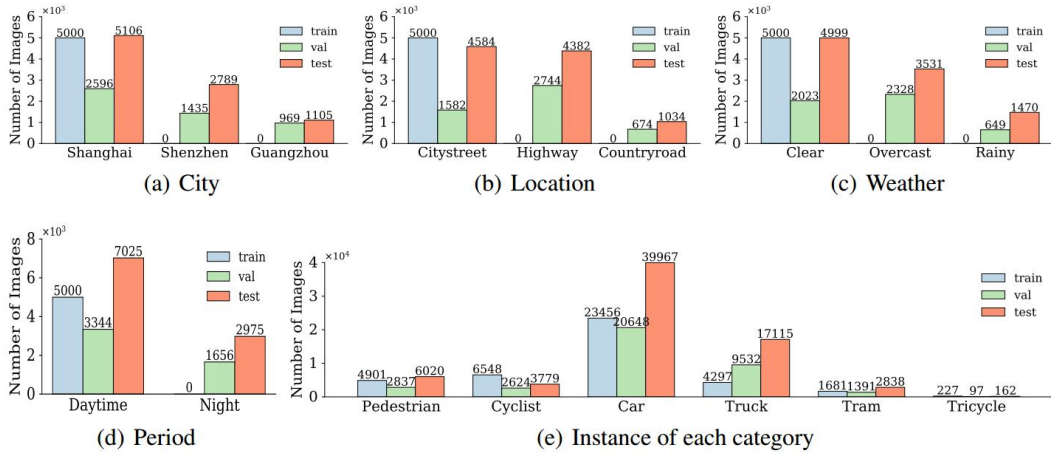


图 28 有标签数据集统计信息(a)每个城市的图像数目(b)每个地点的图像数目(c)每种天气情况下的图像数目(d)每个时期的图像数量(e)每一类案件的数目。

由于 SODA10M 是一个全新的自动驾驶数据集，其在基于几个具有代表性的一级和二级检测器上提供了完全监督的基线结果。此外，在大量未标记数据中，SODA10M 还精心挑选了最具代表性的自监督和半监督方法，并研究了这些方法在 SODA10M 上的泛化能力。图 29 展示了 SODA10M 团队在这些方法上的研究成果。

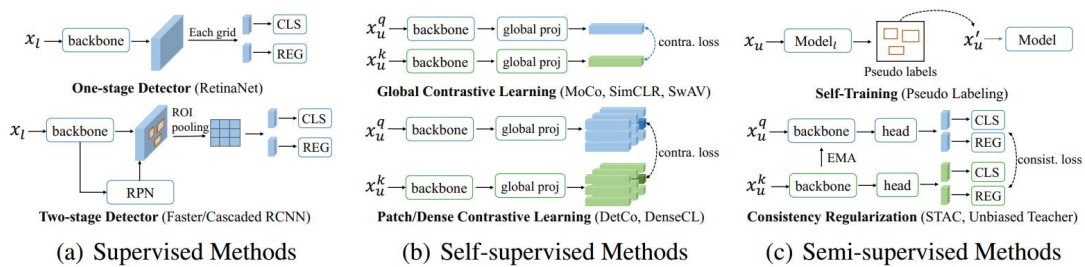


图 29 构建 SODA10M 基准的不同方法。

SODA10M 是一个以自监督和半监督学习为重点的大规模 2D 自动驾驶数据集，提供了大量的未标记数据和一小部分高质量的标记数据。这些数据来自不同天气条件、时期和位置场景下的各个城市。与现有的自动驾驶数据集相比，SODA10M 的规模最大，数据更加多样化。此外，SODA10M 建立了自动驾驶场景下的自监督和半监督学习基准，表明其可以作为一个有前途的数据集，用于训练和评估不同的自监督和半监督学习方法。

1.2 自动驾驶数据集数据集比较

1.2.1 数据集特征对比分析

在上述章节我主要介绍了 KITTI、Waymo、PandaSet、Apollo 和 SODA10M 这五种数据集，为了方便对比我将其各自的特征做成了表格的形式，如表 1 所示，其中从数据集名称、数据集来源、来源地区、发表时间、采集地点、环境特征、道路特征、传感器、标注种类、是否含有目标追踪、帧数、标注类别 12 个角度进行对比。

数据集名称	KITTI	Waymo	PandaSet	Apollo Scape	SODA10M
数据集来源	德国卡尔斯鲁厄理工学院(KIT)&丰田工业大学芝加哥分校(TTIC)	Waymo Inc	禾赛科技 &Scale AI	百度	华为诺亚方舟实验室&中山大学
来源地区	国外	国外	国内外合作	中国	中国
发布时间	2021	2019-2021	2019	2018-2020	2021
采集地点	卡尔斯鲁厄	美国	硅谷	中国北京	中国 32 个城市
环境特征	/	雨/夜间/住宅区/乡村	/	不同照明调剂	晴/阴/雨/夜间
道路特征	城市/高速/住宅区/乡村/园区	城市/高速/住宅区/乡村	/	城市	城市/高速/住宅区/乡村
传感器	摄像头 /LiDAR/GPS	摄像头 /LiDAR/GPS	摄像头 /LiDAR/GPS	摄像头/LiDAR	行车记录仪
标注种类	2D/3D	2D/3D	2D/3D	2D/3D	2D
目标追踪	有	有	有	有	无
帧数(k)	15	200	60	110	/
标注类别	2	4	28	10	6

表 1 本文所介绍的五种数据集对比

这些自动驾驶数据集在多个方面展示了各自独特的特征和来源背景。KITTI 数据集源于德国卡尔斯鲁厄理工学院和丰田工业大学芝加哥分校的合作，主要采集于欧洲，以城市、高速公路、住宅区和乡村等不同道路环境为特征，传感器包括摄像头、LiDAR 和 GPS，支持 2D 和 3D 标注，并实现目标追踪功能。Waymo 数据集由美国的 Waymo 公司发布，覆盖美国的城市和高速公路，标注种类为 2D

和 3D，同样支持目标追踪。PandaSet 数据集是由禾赛科技和 Scale AI 合作发布，采集地点涵盖硅谷和中国，传感器包括摄像头、LiDAR 和 GPS，标注种类较多，达到 28 种，同样支持目标追踪功能。Apollo Scape 数据集则是由中国北京和华为诺亚方舟实验室合作发布，环境特征包括不同的照明调剂和城市道路，传感器为摄像头和 LiDAR，标注种类为 2D 和 3D，标注类别数量较多。最后，SODA10M 数据集由华为和中山大学合作，采集于中国 32 个城市，传感器为行车记录仪，主要用于 2D 标注，不支持目标追踪功能。这些数据集的多样性和特征差异，为自动驾驶算法和系统的开发提供了丰富的基础和资源。

1.2.2 论文数据集对比

除了上述数据集的详细特征描述，我在查阅相关论文时还注意到了一些关于不同数据集之间比较的内容。在这部分，我将对论文中的数据集比较部分进行概述和汇总。

在论文 *Scalability in Perception for Autonomous Driving: Waymo Open Dataset* [2]中，KITTI 对一些流行的数据集进行了比较。其中 Argo 数据集主要涉及他们的跟踪数据集，而非运动预测数据集。在投影到 2D 的情况下，3D 标签不计算在 2D 框内。平均点/帧是指所有 LiDAR 返回的点的数量，这些点是根据采集到的数据计算得出的。通过在半径 75 米内对轨迹进行稀疏处理，并合并所有稀疏区域，来测量访问面积。

关键观察如下：首先，KITTI 数据集在有效地理覆盖方面是其他数据集的 15.2 倍。其次，KITTI 数据集相比其他相机与激光雷达数据集更为广泛。具体不同数据集之间的比较见表 2。这些比较的结果有助于研究人员了解各个数据集的特点和适用场景，为他们选择合适的数据集进行特定任务提供了重要参考。

	KITTI	NuScenes	Argo	Ours
Scenes	22	1000	113	1150
Ann. Lidar Fr.	15K	40K	22K	230K
Hours	1.5	5.5	1	6.4
3D Boxes	80K	1.4M	993k	12M
2D Boxes	80K	–	–	9.9M
Lidars	1	1	2	5
Cameras	4	6	9	5
Avg Points/Frame	120K	34K	107K	177K
LiDAR Features	1	1	1	2
Maps	No	Yes	Yes	No
Visited Area (km ²)	–	5	1.6	76

表 2 KITTI 与其他数据集对比

在 *The ApolloScape Open Dataset for Autonomous Driving and its Application* [4] 论文中, 比较了 Apollo 数据集和其他 SOTA 自动驾驶数据集的属性, 并表明 Apollo 在数据规模、标签粒度、真实环境中的任务变化方面是独一无二的。Apollo 数据集与其他已发布的街景自动驾驶数据集的比较。具体展示如表 3 所示。

Dataset	Real	Location Accuracy	Diversity	Annotation			
				3D	2D	Video	Lane
CamVid [26]	✓	-	day time	no	pixel: 701	✓	2D / 2 classes
Kitti [2]	✓	cm	day time	80k 3D box	box: 15k pixel: 400	-	no
Cityscapes [3]	✓	-	day time 50 cities	no	pixel: 25k	-	no
Toronto [27]	✓	cm	Toronto	focus on buildings and roads exact numbers are not available ¹			
Mapillary [28]	✓	meter	various weather day & night 6 continents	no	pixel: 25k	-	2D / 2 classes
BDD100K [29]	✓	meter	various weather day 4 regions in US	no	box: 100k pixel: 10k	-	2D / 2 classes
SYNTHIA [30]	-	-	various weather	box	pixel: 213k	✓	no
P.F.B. [31]	-	-	various weather	box	pixel: 250k	✓	no
ApolloScape	✓	cm	various weather day time 4 regions in China	3D semantic point 70K 3D fitted cars	pixel: 140k	✓	3D / 2D Video 27 classes

表 3 Apollo 与其他数据集对比。“pixel”表示二维像素级注释。“point”表示 3D 点级注释。“box”表示边框级别的注释。“Video”表示是否对 2D 视频序列进行标注。

在这篇科研论文 *SODA10M: A Large-Scale 2D Self/Semi-Supervised Object Detection Dataset for Autonomous Driving* [7] 中, 数据集统计与现有的自动驾驶数据集进行了比较, 具体见表 4。SODA10M 显著超过了现有的 BDD100K、Waymo 等自动驾驶数据集。SODA10M 包含 1000 万张道路场景图像, 即使采集间隔更长 (每帧 10 秒), 其数量也是 Waymo 的 10 倍。在评估不同的自监督和半监督学习方法时, 我们主要考虑数据集中图像的数量, 而不是标记的图像。

在表 4 中, “夜/雨”一栏指示数据集是否包含与夜间或雨天场景相关的域信息。“视频”则表示数据集是否提供视频格式或详细的时间顺序信息。这些统计数据为研究人员提供了选择适合特定研究目的的数据集时的参考依据。

Dataset	Images	Cities	Night/Rain	Video	Driving hours	Resolution
Caltech Pedestrian [11]	249K	5	X/X	✓	10	640×480
KITTI [16]	15K	1	X/X	X	6	1242×375
Citypersons [71]	5K	27	X/X	X	-	2048×1024
BDD100K [68]	100K	4	✓/✓	✓	1111	1280×720
nuScenes [1]	1.4M	2	✓/✓	✓	5.5	1600×900
Waymo Open [51]	1M	3	✓/✓	✓	6.4	1920×1280
SODA10M (Ours)	10M	32	✓/✓	✓	27833	1920×1080

表 4 Apollo 与其他数据集对比

1.3 街景三维重建任务论述

由于子任务 2 是对神经渲染的街景三维重建算法进行具体的阐述，所以本部分主要是概况了街景三维重建任务的比较普适的四个过程，但是具体到每个方法可能会稍有不同，对于每个过程中的详细原理和关键技术我将在子任务 2 中进行详细是编写。

1.3.1 数据处理

下载数据集如图 30，并解压数据集到指定地点，这里的数据集可以是上述我介绍的五种，也可以是其他相关数据集，当然自己构建数据集也可以。一般项目需要的数据集的结构可能会与解压完数据集的结构不太相同，这时候我们可能需要一些脚本语言去重构数据集的格式，如图 31。








 KITTI_02_140_f85_qd3dt.zip	2024/7/10 15:01	WinRAR ZIP 压缩文件	649,114 KB
 KITTI_06_65_f56_qd3dt.zip	2024/7/10 15:01	WinRAR ZIP 压缩文件	687,249 KB
 KITTI360_0000_5200_5266.zip	2024/7/10 15:07	WinRAR ZIP 压缩文件	1,615,971 KB
 KITTI360_0000_5335_5415.zip	2024/7/10 15:06	WinRAR ZIP 压缩文件	1,498,939 KB
 KITTI360_0000_5954_6034.zip	2024/7/10 15:07	WinRAR ZIP 压缩文件	1,751,918 KB
 Waymo_176124_0_102.zip	2024/7/10 15:08	WinRAR ZIP 压缩文件	2,942,473 KB

图 30 下载数据集

```
clip root
|-- images
|   |-- ${cam0 name}
|   |   |-- t0.jpg/png
|   |   |-- t1.jpg/png
|   |   |-- ...
|   |-- ${cam1 name}
|   |-- ...
|-- lidars
|   |-- ${lidar0 name}
|   |   |-- t0.pcd
|   |   |-- t1.pcd
|   |   |-- ...
|   |-- ${lidar1 name}
|   |-- ...
|-- transform.json
|-- annotation.json
```













-  aggregate_lidar 2024/7/10 20:21
-  colmap 2024/7/10 20:21
-  images 2024/7/10 20:21
-  lidars 2024/7/10 20:21
-  masks 2024/7/10 20:21
-  masks_box2d 2024/7/10 20:21
-  segs 2024/7/10 20:21
-  .DS_Store 2024/5/28 11:54
-  annotation.json 2024/5/28 11:55
-  transform.json 2024/5/28 12:17

图 31 数据处理一

还有一种情况是比如我们通过录视频的方式来作为数据源，我们可以直接编写代码将视频分解为图片并保存到相应的文件夹下，如图 32 所示。

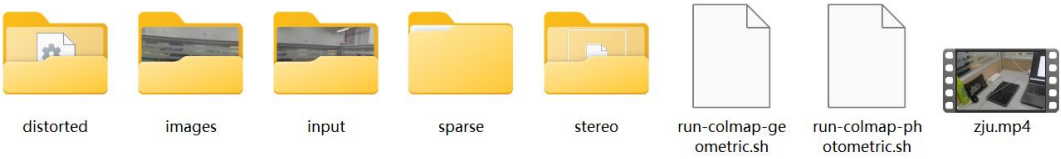


图 32 数据处理二

在进行上述步骤之后，数据还需进行预处理，包括图像预处理和点云预处理。图像预处理包括去噪、色彩校正、畸变校正等步骤，以确保图像的质量和一致性。点云预处理则涉及滤波、降噪和分割，目的是去除冗余和噪声点，得到干净的点云数据。接着，还需要进行同步与校准，对图像和点云数据进行时间和空间上的同步与校准，以保证多视角数据的一致性。

在完成这些预处理工作之后，还需进行特征提取与匹配。需要使用特征点提取算法（如 **SIFT**、**SURF**）从图像中提取关键特征点，并进行特征匹配，以找到图像间的对应关系。

1.3.2 三维重现

在街景三维重建任务中，选择适当的三维重建算法至关重要。传统的三维创建算法有基于立体视觉的重建方法、激光扫描、基于特征点的重建等方法。而现代神经渲染技术的代表算法如 **NeRF** 和 **3D Gaussian** 在街景三维重建领域备受关注。**NeRF** 通过密集采样和神经网络来建模场景的复杂光学属性，而 **3D Gaussian** 算法则通过将三维点云视为高斯分布的集合来进行初步模型初始化。

在街景三维重建任务的普遍过程中，首先是结构从运动（**SfM**）阶段。这一步骤利用从图像中提取的特征点及它们的匹配关系，估算出相机在空间中的位置和姿态。这些信息生成了一个稀疏的三维点云，描述了场景中相机和特征点的大致空间位置。

接下来是多视图立体（**MVS**）阶段，它进一步处理多个视角的图像，以生成更为密集和精确的三维点云。通过综合多个角度的观察和相机参数，结合特征点的深度信息，**MVS** 能够生成具有高精度和细节的三维重建结果。

最后是模型初始化阶段，其中初步生成的三维点云被视为一组高斯分布。每个点云点通过其位置和颜色信息确定了初始的高斯分布参数，这种方法有助于后续的三维重建和数据处理工作，如物体识别和场景分析等。

在 **3D Gaussian** 算法的三维重建阶段，整合后的数据用于生成真实世界的三维场景。这包括建立建筑物、道路网络、绿地和水体等基本几何结构，并确保它们在空间中的准确位置和比例关系。在此过程中，常用的算法涉及从点云数据到几何网格的转换，以及从图像到纹理贴图的映射，以增强场景的视觉真实感。

1.3.3 模型优化

在三维重建的迭代优化过程中，首先需要定义一个完备的损失函数，用于评估模型与真实数据之间的差异。这个损失函数通常包括重建误差和正则化项，重建误差衡量了模型预测与真实数据之间的差异，而正则化项则有助于控制模型的

复杂度，避免过拟合。

接下来，通过梯度下降等优化方法，对所选模型的参数进行迭代更新，以最小化定义的损失函数。梯度下降算法通过计算损失函数对参数的梯度，并沿着梯度的反方向更新参数，使得模型能够逐步优化，提高与真实数据的拟合度。

在优化的过程中，还可以采用神经渲染技术。通过神经网络生成合成图像，并将其与真实图像进行比较，可以实时地调整模型参数，以提高模型的精度和细节。神经渲染技术利用深度学习模型对场景的光学特性进行建模，能够生成高质量的合成图像，进一步指导优化过程。

随着迭代优化的进行，还会生成更为密集的三维点云如图 33 所示。密集点云能够捕捉更多的细节信息，提高模型的真实性和视觉效果。通过增加点云的密度，可以更准确地表达场景的空间结构和表面细节，从而实现更高质量的三维重建结果。图 34 在虚幻引擎中展示了不同训练轮数下点云对比，其中 30000 轮下的点云相比于 7000 轮下的点云明显细节更加丰富，质量更高。

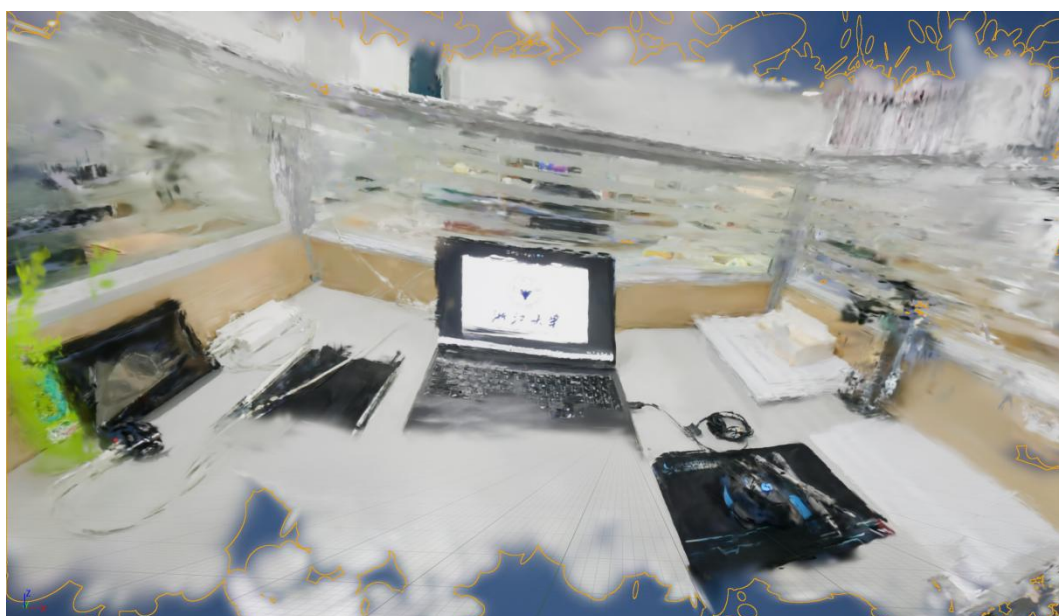


图 33 训练 7000 轮点云效果



图 34 训练 7000 轮和 30000 轮点云对比（黄色高亮是训练 30000 轮的点云）

1.3.4 评估与验证

在对三维重建模型进行评估时，我们首先进行视觉评估。这一步骤涉及渲染合成图像并将其与实际街景图像进行对比，以评估模型的逼真度和对细节的还原能力。视觉评估能够直观地检查模型在视觉上的表现，帮助确认其在模拟真实场景方面的效果。

其次是定量评估，采用常见的三维重建评估指标，如重建误差、点云密度以及纹理质量等，对模型进行定量分析。这些指标可以量化地评估模型在几何精度和视觉质量上的表现，为进一步改进提供数据支持。

在模型验证阶段，我们采用交叉验证的方法，将数据集分为训练集和验证集，以评估模型的泛化能力和鲁棒性。此外，通过使用额外的街景数据进行外部测试，确保模型能够在不同环境和条件下保持高质量的重建效果，验证其在实际应用中的可靠性和稳健性。

PSNR、SSIM 和 LPIPS 是常用的三种评价指标。PSNR（峰值信噪比）用于评估图像重建的保真度，通过比较原始图像和重建图像的均方误差来衡量。SSIM（结构相似性指数）则用于评估两幅图像之间的结构相似性，考虑了亮度、对比度和结构信息的影响。而 LPIPS（学习感知图像块相似性）是基于深度学习的评价指标，更加接近人类视觉系统的感知特征，用于评估图像的质量和相似性。图 35 是在复现论文 *HUGS: Holistic Urban 3D Scene Understanding via Gaussian Splatting* [8]时得到的模型训练评估参数。

```
Metric evaluation progress: 100%|
SSIM :    0.9131930
PSNR :    26.6772747
LPIPS:    0.0325859

Method: ours_30000
Metric evaluation progress: 100%|
SSIM :    0.9207560
PSNR :    27.7937775
LPIPS:    0.0285865
```

图 35 模型训练评估

二、附录

以下代码均在 GitHub 上开源，开源地址如下。

(<https://github.com/Metastarx/ZJU>)

```
1. import os
2. import numpy as np
3. import cv2
4. class Calib:
5.     def __init__(self, dict_calib):
6.         super(Calib, self).__init__()
7.         self.P0 = dict_calib['P0'].reshape(3, 4)
8.         self.P1 = dict_calib['P1'].reshape(3, 4)
9.         self.P2 = dict_calib['P2'].reshape(3, 4)
10.        self.P3 = dict_calib['P3'].reshape(3, 4)
11.        self.R0_rect = dict_calib['R0_rect'].reshape(3, 3)
12.        self.P0 = dict_calib['P0'].reshape(3, 4)
13.        self.Tr_velo_to_cam = dict_calib['Tr_velo_to_cam'].reshape(3, 4)
14.        self.Tr_imu_to_velo = dict_calib['Tr_imu_to_velo'].reshape(3, 4)
15.
16.
17. class Object3d:
18.     def __init__(self, content):
19.         super(Object3d, self).__init__()
20.         # content 就是一个字符串，根据空格分隔开来
21.         lines = content.split()
22.
23.         # 去掉空字符
24.         lines = list(filter(lambda x: len(x), lines))
25.
26.         self.name, self.truncated, self.occluded, self.alpha = lines[0], float(lines[1]), float(lines[2]), float(lines[3])
27.
28.         self.bbox = [lines[4], lines[5], lines[6], lines[7]]
29.         self.bbox = np.array([float(x) for x in self.bbox])
30.         self.dimensions = [lines[8], lines[9], lines[10]]
31.         self.dimensions = np.array([float(x) for x in self.dimensions])
32.         self.location = [lines[11], lines[12], lines[13]]
33.         self.location = np.array([float(x) for x in self.location])
34.         self.rotation_y = float(lines[14])
35.         #这一行是模型训练后的 label 通常最后一行是阈值，可以同个这个过滤掉概率低的 object
```

```

36.         #如果只要显示 kitti 本身则不需要这一行
37.         #self.ioc = float(lines[15])
38.
39.
40.     class Kitti_Dataset:
41.         def __init__(self, dir_path, split="training"):
42.             super(Kitti_Dataset, self).__init__()
43.             self.dir_path = os.path.join(dir_path, split)
44.             # calib 矫正参数文件夹地址
45.             self.calib = os.path.join(self.dir_path, "calib")
46.             # RGB 图像的文件夹地址
47.             self.images = os.path.join(self.dir_path, "image_2")
48.             # 点云图像文件夹地址
49.             self.pcs = os.path.join(self.dir_path, "velodyne")
50.             # 标签文件夹的地址
51.             self.labels = os.path.join(self.dir_path, "label_2")
52.
53.         # 得到当前数据集的大小
54.         def __len__(self):
55.             file = []
56.             for _, _, file in os.walk(self.images):
57.                 pass
58.
59.             # 返回 rgb 图片的数量
60.             return len(file)
61.
62.         # 得到矫正参数的信息
63.         def get_calib(self, index):
64.             # 得到矫正参数文件
65.             calib_path = os.path.join(self.calib, "{:06d}.txt".format(index))
66.             with open(calib_path) as f:
67.                 lines = f.readlines()
68.
69.             lines = list(filter(lambda x: len(x) and x != '\n', lines))
70.             dict_calib = {}
71.             for line in lines:
72.                 key, value = line.split(":")
73.                 dict_calib[key] = np.array([float(x) for x in value.split()])
74.             return Calib(dict_calib)
75.
76.         def get_rgb(self, index):
77.             # 首先得到图片的地址
78.             img_path = os.path.join(self.images, "{:06d}.png".format(index))
79.             return cv2.imread(img_path)

```



```

80.
81.     def get_pcs(self, index):
82.         pcs_path = os.path.join(self.pcs, "{:06d}.bin".format(index))
83.         # 点云的四个数据 (x, y, z, r)
84.         aaa = np.fromfile(pcs_path, dtype=np.float32, count=-1).reshape([-1, 4])
85.         return aaa[:, :3]
86.
87.     def get_labels(self, index):
88.         labels_path = os.path.join(self.labels, "{:06d}.txt".format(index))
89.         with open(labels_path) as f:
90.             lines = f.readlines()
91.             lines = list(filter(lambda x: len(x) > 0 and x != '\n', lines))
92.
93.         return [Object3d(x) for x in lines]

```

代码 1 kitti_Dataset.py

```

1.     import os
2.
3.     import cv2
4.     import numpy as np
5.     import time
6.     import open3d as o3d
7.     from data.kitti_Dataset import *
8.
9.     # 根据偏航角计算旋转矩阵（逆时针旋转）
10.    def rot_y(rotation_y):
11.        cos = np.cos(rotation_y)
12.        sin = np.sin(rotation_y)
13.        R = np.array([[cos, 0, sin], [0, 1, 0], [-sin, 0, cos]])
14.        return R
15.    def draw_3dframeworks(vis, points):
16.
17.        position = points
18.        points_box = np.transpose(position)
19.
20.        lines_box = np.array([[0, 1], [1, 2], [0, 3], [2, 3], [4, 5], [4, 7], [5, 6], [6, 7],
21.                               [0, 4], [1, 5], [2, 6], [3, 7], [0, 5], [1, 4]])
22.        colors = np.array([[1., 0., 0.] for j in range(len(lines_box))])
23.        line_set = o3d.geometry.LineSet()
24.
25.        line_set.points = o3d.utility.Vector3dVector(points_box)
26.        line_set.lines = o3d.utility.Vector2iVector(lines_box)
27.        line_set.colors = o3d.utility.Vector3dVector(colors)

```

```

28.
29.
30.     render_option.line_width = 5.0
31.     vis.update_geometry(line_set)
32.     render_option.background_color = np.asarray([1, 1, 1])
33.     # vis.get_render_option().load_from_json('renderoption_1.json')
34.     render_option.point_size = 4
35.     #param = o3d.io.read_pinhole_camera_parameters('BV.json')
36.
37.
38.
39.     print(render_option.line_width)
40.     ctr = vis.get_view_control()
41.
42.     vis.add_geometry(line_set)
43.     #ctr.convert_from_pinhole_camera_parameters(param)
44.     vis.update_geometry(line_set)
45.     vis.update_renderer()
46.
47.     if __name__ == "__main__":
48.         dir_path = "data/kitti/"
49.         index = 10 #图片的标号
50.         split = "training"
51.         dataset = Kitti_Dataset(dir_path, split=split)
52.
53.         vis = o3d.visualization.Visualizer()
54.         vis.create_window(width=771, height=867)
55.
56.         obj = dataset.get_labels(index)
57.         img3_d = dataset.get_rgb(index)
58.         calib1 = dataset.get_calib(index)
59.         pc = dataset.get_pcs(index)
60.         print(img3_d.shape)
61.         point_cloud = o3d.geometry.PointCloud()
62.
63.         point_cloud.points = o3d.utility.Vector3dVector(pc)
64.         point_cloud.paint_uniform_color([0, 121/255, 89/255])
65.         vis.add_geometry(point_cloud)
66.         render_option = vis.get_render_option()
67.         render_option.line_width = 4
68.
69.         for obj_index in range(len(obj)):
70.             if obj[obj_index].name == "Car" or obj[obj_index].name == "Pedestrian" or obj[obj_index].name == "Cyclist":

```

```

71.         # 阈值设置 ioc
72.         # 如果需要显示自己的 traininglabel 结果，需要取消这样的注释，并取消 object3d.py 最后一
    行的注释
73.         #if (obj[obj_index].name == "Car" and obj[obj_index].ioc >= 0.7) or obj[obj_in
    dex].ioc > 0.5:
74.             R = rot_y(obj[obj_index].rotation_y)
75.             h, w, l = obj[obj_index].dimensions[0], obj[obj_index].dimensions[1], obj[o
    bj_index].dimensions[2]
76.             x = [l / 2, l / 2, -l / 2, -l / 2, l / 2, l / 2, -l / 2, -l / 2]
77.             y = [0, 0, 0, 0, -h, -h, -h, -h]
78.             # y = [h / 2, h / 2, h / 2, h / 2, -h / 2, -h / 2, -h / 2, -h / 2]
79.             z = [w / 2, -w / 2, -w / 2, w / 2, w / 2, -w / 2, -w / 2, w / 2]
80.             # 得到目标物体经过旋转之后的实际尺寸（得到其在相机坐标系下的实际尺寸）
81.             corner_3d = np.vstack([x, y, z])
82.             corner_3d = np.dot(R, corner_3d)
83.
84.             # 将该物体移动到相机坐标系下的原点处（涉及到坐标的移动，直接相加就行）
85.             corner_3d[0, :] += obj[obj_index].location[0]
86.             corner_3d[1, :] += obj[obj_index].location[1]
87.             corner_3d[2, :] += obj[obj_index].location[2]
88.             corner_3d = np.vstack((corner_3d, np.zeros((1, corner_3d.shape[-1]))))
89.             corner_3d[-1][-1] = 1
90.
91.
92.             inv_Tr = np.zeros_like(calib1.Tr_velo_to_cam)
93.             inv_Tr[0:3, 0:3] = np.transpose(calib1.Tr_velo_to_cam[0:3, 0:3])
94.             inv_Tr[0:3, 3] = np.dot(-np.transpose(calib1.Tr_velo_to_cam[0:3, 0:3]), cal
    ib1.Tr_velo_to_cam[0:3, 3])
95.
96.             Y = np.dot(inv_Tr, corner_3d)
97.
98.             draw_3dframeworks(vis, Y)
99.
100.         vis.run()

```

代码 2 draw_3d.py

```

1.     import cv2
2.     import numpy as np
3.     from data.kitti_Dataset import *
4.
5.     # 根据偏航角计算旋转矩阵（逆时针旋转）
6.     def rot_y(rotation_y):
7.         cos = np.cos(rotation_y)
8.         sin = np.sin(rotation_y)

```

```

9.         R = np.array([[cos, 0, sin], [0, 1, 0], [-sin, 0, cos]])
10.     return R
11.
12.     if __name__ == "__main__":
13.         # 读取的数据的文件夹
14.
15.         dir_path = "data/kitti/"
16.         #index = 10 #图片的标号
17.         split = "training"
18.         dataset = Kitti_Dataset(dir_path, split=split)
19.
20.
21.         k = 10
22.         img3_d = dataset.get_rgb(k)
23.         print(img3_d.shape)
24.         max_num = 100
25.         # 逐张读入图片
26.         while True:
27.
28.
29.             img3_d = dataset.get_rgb(k)
30.
31.             calib = dataset.get_calib(k)
32.
33.             # 获取标签数据
34.             obj = dataset.get_labels(k)
35.
36.             # 逐个读入一副图片中的所有 object 的标签
37.             for num in range(len(obj)):
38.                 if obj[num].name == "Car" or obj[num].name == "Pedestrian" or obj[num].name ==
"Cyclist":
39.                     #这一行为阈值用来过滤训练概率较低的 object
40.                     #if (obj[num].name == "Car" and obj[num].ioc >= 0.7) or obj[num].ioc > 0.5:
41.
42.                     # step1 得到 rot_y 旋转矩阵 3*3
43.                     R = rot_y(obj[num].rotation_y)
44.                     # 读取 object 物体的高宽长信息
45.                     h, w, l = obj[num].dimensions[0], obj[num].dimensions[1], obj[num].dimensions[2]
46.
47.                     # step2
48.                     # 得到该物体的坐标以底面为原点中心所在的物体坐标系下各个点的坐标
49.                     #      7 ----- 4

```



```

50.          #   /|           /|
51.          #   6  -----  5  .
52.          #   | |           | |
53.          #   . 3  -----  0
54.          #   | /   .- - -| / - - -> (x)
55.          #   2  ---|----- 1
56.          #           |
57.          #           | (y)
58.          x = [1 / 2, 1 / 2, -1 / 2, -1 / 2, 1 / 2, 1 / 2, -1 / 2, -1 / 2]
59.          y = [0, 0, 0, 0, -h, -h, -h, -h]
60.          z = [w / 2, -w / 2, -w / 2, w / 2, w / 2, -w / 2, -w / 2, w / 2]
61.          # 将xyz 转化成 3*8 的矩阵
62.          corner_3d = np.vstack([x, y, z])
63.          # R * X
64.          corner_3d = np.dot(R, corner_3d)
65.
66.          # 将该物体移动到相机坐标系下的原点处（涉及到坐标的移动，直接相加就行）
67.          corner_3d[0, :] += obj[num].location[0]
68.          corner_3d[1, :] += obj[num].location[1]
69.          corner_3d[2, :] += obj[num].location[2]
70.
71.          # 将 3d 的 bbox 转换到 2d 坐标系中（需要用到内参矩阵）
72.          corner_3d = np.vstack((corner_3d, np.zeros((1, corner_3d.shape[-1]))))
73.
74.          corner_2d = np.dot(calib.P2, corner_3d)
75.          # 在像素坐标系下，横坐标 x = corner_2d[0, :] /= corner_2d[2, :]
76.          # 纵坐标的值以此类推
77.          corner_2d[0, :] /= corner_2d[2, :]
78.          corner_2d[1, :] /= corner_2d[2, :]
79.
80.          corner_2d = np.array(corner_2d, dtype=np.int)
81.          # 绘制立方体边界框
82.          color = [0, 255, 0]
83.          # 线宽
84.          thickness = 2
85.
86.          #绘制 3d 框
87.          for corner_i in range(0, 4):
88.              i, j = corner_i, (corner_i + 1) % 4
89.              cv2.line(img3_d, (corner_2d[0, i], corner_2d[1, i]), (corner_2d[0,
90.                  j], corner_2d[1, j]), color, thickness)
91.              i, j = corner_i + 4, (corner_i + 1) % 4 + 4

```

```

91.                cv2.line(img3_d, (corner_2d[0, i], corner_2d[1, i]), (corner_2d[0,
                j], corner_2d[1, j]), color, thickness)
92.                i, j = corner_i, corner_i + 4
93.                cv2.line(img3_d, (corner_2d[0, i], corner_2d[1, i]), (corner_2d[0,
                j], corner_2d[1, j]), color, thickness)
94.
95.
96.                cv2.line(img3_d,(corner_2d[0, 0],corner_2d[1, 0]), (corner_2d[0, 5], co
                rner_2d[1, 5]),color, thickness)
97.                cv2.line(img3_d, (corner_2d[0, 1], corner_2d[1, 1]), (corner_2d[0, 4],
                corner_2d[1, 4]), color, thickness)
98.                cv2.imshow("{}".format(k), img3_d, )
99.                cv2.moveWindow({}, 300, 50)
100.               key = cv2.waitKey(100) & 0xFF
101.               if key == ord('d'):
102.                   k += 1
103.                   cv2.destroyAllWindows()
104.                   # if idx == 104:
105.                       #     idx += 1
106.               if key == ord('a'):
107.                   k -= 1
108.
109.               if key == ord('q'):
110.                   break
111.               if k >= max_num:
112.                   k = max_num - 1
113.               if k < 0:
114.                   k = 0
115.               # 读入图片信息

```

代码 3 img_3dbbox.py

三、引用

- [1] Geiger A, Lenz P, Stiller C, Urtasun R. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*. 2013;32(11):1231-1237.
- [2] Sun, P., Kretschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., ... Angelov, D. (2020). Scalability in Perception for Autonomous Driving: Waymo Open Dataset.
- [3] Yan, Y., Lin, H., Zhou, C., Wang, W., Sun, H., Zhan, K., ... Peng, S. (2024). Street Gaussians for Modeling Dynamic Urban Scenes.
- [4] Wang, P., Huang, X., Cheng, X., Zhou, D., Geng, Q., & Yang, R. (2019). The apolloscape open dataset for autonomous driving and its application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- [5] Ma, Y., Zhu, X., Zhang, S., Yang, R., Wang, W., & Manocha, D. (2019). TrafficPredict: Trajectory Prediction for Heterogeneous Traffic-Agents.
- [6] Liao, M., Lu, F., Zhou, D., Zhang, S., Li, W., & Yang, R. (2020). DVI: Depth Guided Video Inpainting for Autonomous Driving.
- [7] Han, J., Liang, X., Xu, H., Chen, K., Hong, L., Mao, J., ... Xu, C. (2021). SODA10M: A Large-Scale 2D Self/Semi-Supervised Object Detection Dataset for Autonomous Driving. *arXiv preprint arXiv:2106.11118*.
- [8] Zhou, H., Shao, J., Xu, L., Bai, D., Qiu, W., Liu, B., ... Liao, Y. (2024). HUGS: Holistic Urban 3D Scene Understanding via Gaussian Splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 21336-21345).