All-In-One Drive: A Comprehensive Perception Dataset with High-Density Long-Range Point Clouds Supplementary Materials

Xinshuo Weng, Yunze Man, Jinhyung Park, Ye Yuan, Matthew O'Toole, Kris Kitani
Robotics Institute, Carnegie Mellon University
{xinshuow, yman, jinhyun1, yyuan2, motoole2, kkitani}@cs.cmu.edu

Abstract

In this supplementary material, we provide details including our data generation process, baseline implementation details, data visualization, and justification to our dataset design, which are not described in the main paper due to limited space.

1 Date simulation procedure

For each scene in our dataset, we choose one of eight cities from Carla assets and sample locations covering most parts of the city to generate agents. For each agent (vehicles, people), we set a random and faraway target destination to generate diverse trajectories. We randomly customize the behavior (*e.g.*, maximum speed, how often to ignore red light, how often to cross the road) for each agent to increase the diversity of the data. Once the environment is set up, we randomly select a vehicle as our ego-vehicle and equip our full sensor suite to this vehicle for data recording. For agents who have approached their destinations, we add another faraway destination to them so that there is no dummy agent (except for parked cars) in our environment. We collected 100 such scenes in our dataset, each containing 1000 frames with full set of annotations.

2 Comprehensive sensor suite

2.1 High-density long-range point clouds

As introduced in the main paper Sec. 3.1, our dataset provides three APD-LiDAR point clouds with a density of 100k, 600k and 1M points per frame, and one depth point cloud with 4M points per frame. Due to limited space in the main paper, we only visualized the 100k LiDAR and 4M depth point clouds. Here, we provide additional comparison between four point clouds in Fig. 1. It is clear to see the difference regarding the density of four point clouds. Also, our 100k LiDAR point cloud, which aims to mimic the specification of the real-world Velodyne-64 [2], has a smaller vertical FoV ($26.9°$) than high-density long-range point clouds ($180°$). Note that, for all our visualizations in the main paper and supplementary, we only show a center crop (up to 120m) of the point clouds, which actually have a range of up to 1000 meters as seen in the main paper Figure 1 (right). This is because showing the full range of the point clouds will lead to the objects too small to see in the figure.

2.2 Depth point cloud generation process

As mentioned in the main paper Sec. 3.1, we provide high-density long-range depth point clouds with 4M points per frame, which are generated by projecting the depth images from five viewpoints to 3D space and then fusing the five point clouds together. To help readers understand the data generation process, we visualize it in Figure 2. Since each of our depth cameras has a sensing range of up to 1000 meters in its direction, our full depth point cloud has a sensing range of 1000 meters in all directions. Note that, as the viewpoint of front left and front right cameras are close to each other (a

Submitted to the 35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks. Do not distribute.

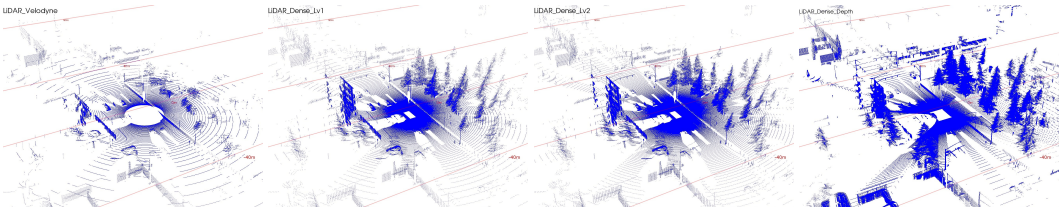

Figure 1: Comparison between our four point clouds with difference densities: three APD-LiDAR point clouds with 100k, 600k and 1M points and one depth point cloud with 4M points per frame.

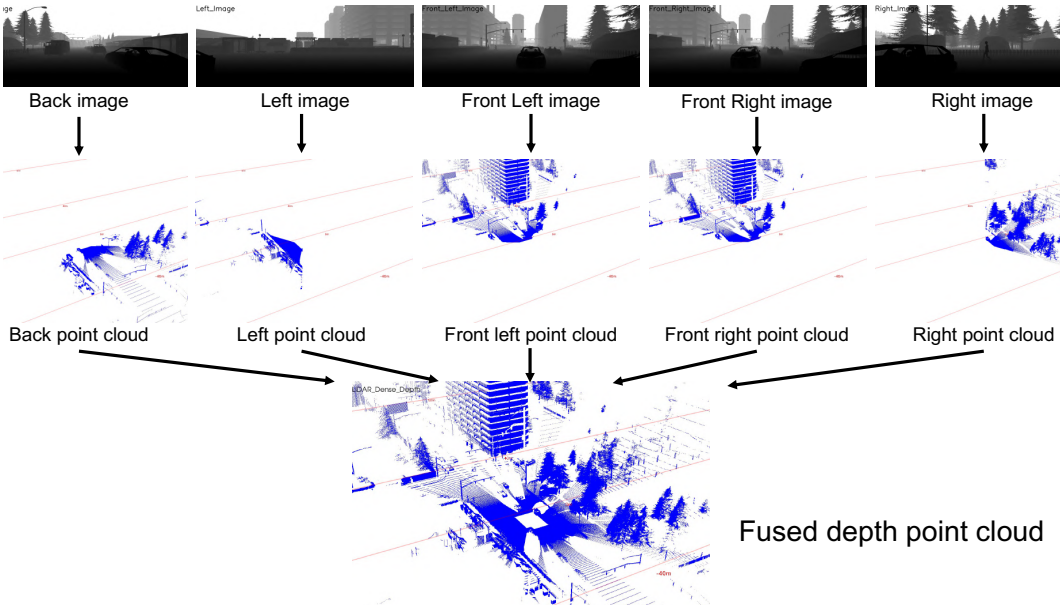

Figure 2: **Depth Point Cloud Generation Process.** Given five depth cameras, we first project the depth image received from the depth cameras to the 3D space to obtain the corresponding point clouds. Then, with five point clouds each covering a viewpoint, we fuse them together to obtain our high-density long-range depth point cloud with 360° horizontal coverage.

large overlapping area), we randomly sample about half the points within the overlapped area so that the point density in all directions is nearly the same.

2.3 SPAD-LiDAR simulation

Though we briefly describe our synthetic SPAD-LiDAR data in the main paper, here we provide details about the sensor simulation process. Our synthetic SPAD-LiDAR algorithm consists of two main components: the simulation of ambient signals (Sec 2.3.2) as well as the simulation of multi-echo (ME) point cloud and reflectance signals (Sec 2.3.3).

2.3.1 Input coordinate transformation

As shown in Algorithm 1, we take RGB, depth and surface normal images, denotes as I_{rgb}, I_d, I_n, as the input of the simulation process, where the RGB and depth signals are directly captured by our RGB and depth cameras, and the surface normal is directly converted from depth images – The process is accurate because of the perfectly synthesized depth values.

In order to mimic the spinning mechanism of LiDAR sensor, we perform a polar-to-image coordinate transformation on all synthesized sensors. Specifically, we approximate the SPAD-LiDAR sensor as a point in the 3D space and define an array of antennas $A[R_V, R_H]$ in the polar coordinate (*i.e.* elevation and azimuth). For all $i \in [1, R_V], j \in [1, R_H]$,

$$A(i,j) = (\theta_i, \gamma_j), \qquad \theta_i, \gamma_j \in [0, 2\pi) \tag{1}$$

where θ and γ are vectors representing the LiDAR sensor detector arrangement, and are predefined according to the desired vertical and horizontal FoV and resolution, and R_H denotes the number of vertically aligned LiDAR detectors on the sensor, and R_V denotes the number of horizontally

Algorithm 1 SPAD-LiDAR Simulation

Inputs: $\mathbf{I_{rgb}}$: RGB image
$\mathbf{I_d}$: Depth image
$\mathbf{I_n}$: Surface normal image

Parameters: $\mathbf{N}$: Number of time bins
$\mathbf{SBR}$: Signal-Background-Ratio
$\mathbf{R_H}, \mathbf{F_H}$: Horizontal Resolution and FoV
$\mathbf{R_V}, \mathbf{F_V}$: Vertical Resolution and FoV
$\mathbf{S}$: Neighborhood aggregation kernel size
$\mathbf{K}$: Number of echo groups

Outputs: $\mathbf{I}_{\mathsf{ambient}}[R_V, R_H]$: Ambient image
$\mathbf{I}_{\mathsf{reflectance}}[R_V, R_H, K]$: Reflectance image
$\mathbf{P_K}$: SPAD-LiDAR Point Cloud Returns

Sample LiDAR sensor array $A[R_H, R_V]$ in the polar coordinate according to FoV F_H, F_V
Project $A[R_H, R_V]$ onto the 2D image space and get the positional map $A_{2D}[R_H, R_V]$
Transform the input images I_{im} into polar coordinate $I_{\mathsf{im}'}$ as Eq. 2 ▷ Get ambient image $\mathbf{I}_{\mathsf{ambient}}[R_V, R_H]$

Calculate signal photons N_{signal} for each pixel (h, w) from Eq. 4;
Calculate ambient photons N_{ambient} for each pixel (h, w) from Eq. 5;
Calculate ambient photons $N_{\mathsf{p}}[R_V, R_H]$ for each pixel (h, w) from Eq. 3;

Simulate multi-echo mechanism and generate $\mathbf{N}^*_{\mathbf{p}}[R_H, R_V, N]$ by neighborhood aggregation as in Eq. 6
Get Top-K returns along the temporal axis with Eq. 7
Project the valid bins into 3D space and generate Top-K point cloud returns ▷ Get multi-echo point cloud $\mathbf{P_K}$
Simulate the reflectance by number of points insides its corresponding bins
Rearrange reflectance values of points into 'LiDAR Image' space ▷ Get reflectance image $\mathbf{I}_{\mathsf{reflectance}}[R_V, R_H, K]$

aligned detection a detector performs during each single sweep. Then, we project the polar coordinate map $A[R_V, R_H]$ onto the 2D image space, resulting in non-linearly arranged positional map $A_{2D}[R_V, R_H]$ on the 2D image plane. Then, we generate new images from the input images by sampling on the original images with the 2D positional map $A_{2D}[R_V, R_H]$. Note that the interpolation is used to handle non-integer positions. For any input image $I_{im} \in \{I_{rgb}, I_d, I_n\}$, $\forall i \in [1, R_V], j \in [1, R_H]$,

$$I'_{im}(i, j) = \text{interp}(I_{im}(x, y)), \;\; (x, y) = A_{2D}(i, j) \tag{2}$$

Then we get new images I'_{rgb}, I'_d, I'_n where pixels are arranged according to the arrangement of LiDAR detector orientations.

2.3.2 Ambient illumination signals

The ambient illumination signals are sunlight reflected by objects. Thus, from the imaging perspective, the information encoded in the ambient signals is very similar to that collected by RGB cameras. Since the LiDAR sensor usually captures light with infrared wavelength, we take R-channel from the RGB images and treat it as a simulation of the ambient signal received by the LiDAR sensor. The ambient signal is represented in the form of images I_{ambient},

$$I_{\mathsf{ambient}} \sim I'_r$$

where I'_r denotes the R-channel of image I' after coordinate transformation.

2.3.3 Multi-echo point cloud and reflectance signals

Generate raw 3D tensor data. Because the LiDAR sensor is essentially a time-of-flight measurement of photons, to simulate multi-signal LiDAR sensing measurements, we first simulate the photon

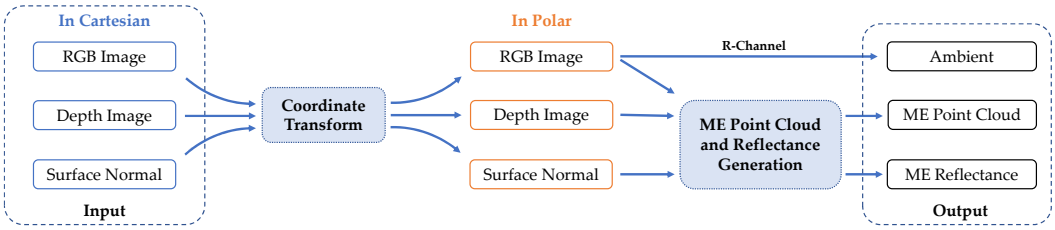

Figure 3: **SPAD-LiDAR measurements simulation.** Taking the inputs from other modalities in the AIODrive Dataset, we first re-sample the images using projected polar positional map to mimic the LiDAR spinning mechanism. Then infrared ambient illumination is approximated by taking R channel of the RGB image. Multi-echo point cloud and reflectance signals are generated from transformed RGB, depth and surface normal images.

measurements. Specifically, without considering the random false detection which happens occasionally in real LiDAR sensors, we formulate the number of photons $\mathbf{N_p}$ received by a LiDAR detector in response to an illumination period of a signal light pulse by a temporal histogram:

$$\mathbf{N_p}[n] \sim \mathcal{P}(N_{\text{signal}}[n] + N_{\text{ambient}}[n]), \tag{3}$$

where n is the n-th time interval along the temporal axis. Function $\mathcal{P}(\cdot)$ models a Poisson distribution, $N_{\text{signal}}[n]$ is the number of detected signal photons at the time interval n, and N_{ambient} models the number of ambient photons. Based on the Eq. 3, the first step of our raw multi-echo LiDAR data generation is to generate a 3D tensor of photon counts $\mathbf{N_p}[R_V, R_H, N]$ representing the number of photons detected by the sensor. Here (R_V, R_H) is the vertical and horizontal resolution of the LiDAR (height and width of the 'LiDAR Image') and N represents the number of time intervals.

To model the number of signal photons $N_{\text{signal}}[n]$, we consider the surface reflectance, angle of incidence during reflection and radial attenuation. We model the relative photon number by assuming all LiDAR transmitters emit lasers with the same energy (same number of photons). Then, according to Lambert's cosine law, the reflected energy is proportional to $\cos(\theta)$ where θ is the incidence angle of the light with respect to the surface. This information is given by surface normal image I'_n. We use a near infrared signal light, *i.e*, the R channel of the RGB image I'_r, to approximate the reflectance of the surface. Also, the radial falloff (attenuation) of light energy is proportional to the square of travel distance. We can directly take advantage of the accurate depth image I'_d. Then, the number of signal photons is modeled as:

$$\mathbf{N}_{\text{signal}}(h,w,n) \sim \begin{cases} \text{Norm}\left(\text{SBR} \times \frac{I'_r(h,w)\cdot\cos\theta}{I'_d(h,w)^2}\right) & \text{If } n = n^* \\ 0 & \text{If } n \neq n^* \end{cases}, \tag{4}$$

where the Norm operation means to normalize over the whole image, (*i.e.* divided by the average value in the entire image), SBR is the Signal-Background-Ratio used to control the relative strength between signal and background light, and n^* is the time bin during which the signal light is reflected by the surface. To model the number of ambient photons $N_{\text{ambient}}[n]$, we simply takes the R-channel of the RGB images and normalize over the whole image,

$$N_{\text{ambient}}(h,w,n) \sim \text{Norm}\left(I_r[h,w]\right), \quad \forall n \in [1,N] \tag{5}$$

Then using Eq. 3 together with Eq. 4 and Eq. 5, we can simulate the 3D tensor of photon number $\mathbf{N_p}[R_V, R_H, N]$. Given the 3D tensor of photon numbers, the next step is to model the multi-echo mechanism. As explained in the main paper, multiple echoes happen because laser beams have a wider coverage of the 3D space instead of a perfect 2D line. In the 2D 'LiDAR Image' space, this can be explained as – neighbor pixels overlap with each other. We use a kernel function $G(\cdot)$ to simulate the spatial coverage, *i.e.*, the number of photons in a given time bin will be a weighted sum of its spatial neighborhood bins (not temporal ones), with nearer neighbors contributing more:

$$\mathbf{N}^*_{\mathbf{p}}[h,w,n] = \sum_{(h,w)\in\mathcal{N}(h,w)}\sum G(k_h,k_w)\cdot\mathbf{N_p}[h,w,n], \tag{6}$$

where $\mathcal{N}$ is the neighborhood of a given position on the image plane, and $G(k_h,k_w)$ is the weight function over the distance between a given 2D position (r_h,r_w) and its neighbor position (k_h,k_w). Specifically, we use a Gaussian function for $G(\cdot)$. By controlling the parameters of the kernel function, we can control the spatial coverage of the lasers.

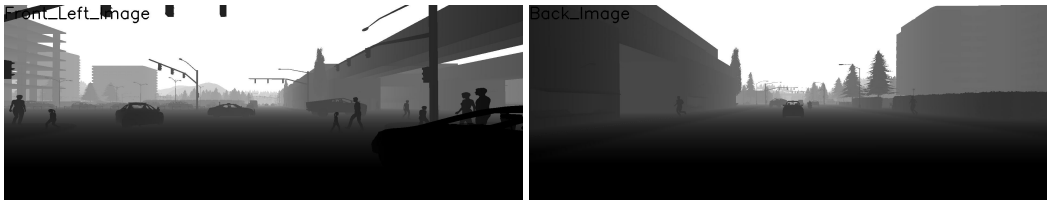

Figure 4: **Depth Images.** We provide visualization of the images captured by depth cameras.

Generate top-K point cloud returns. Because standard LiDAR-based perception systems [3, 4, 1] take point cloud data as input (not raw photon tensor data $\mathbf{N}^*_{\mathbf{p}}[R_V, R_H, N]$), we take one step further to convert our raw multi-echo LiDAR data into point clouds so that they can be easily used in modern perception systems. Note that, with a large spatial coverage rate, each laser beam is able to cover a large 3D volume and is more likely to hit more than one target (object). This is represented as multiple strong peaks along the temporal histogram of a sensor beam. Specifically, when generating the top-K point cloud returns, we take the top-K maximum bins along the temporal axis for each sensor beam, *i.e.*, we select the bin with the top-K number of photons that exceeds a threshold and obtain $\mathbf{N}^*_{\mathbf{p}}[R_V, R_H, K]$ as follows:

$$\mathbf{N}^*_{\mathbf{p}}[R_V, R_H, K] = T\left(S\left(\mathbf{N}^*_{\mathbf{p}}[R_V, R_H, N]\right)[:, :, :\mathrm{K}]\right), \tag{7}$$

where $S(\cdot)$ is a sort function that descendingly sorts the number of photon counts along the temporal axis N. Then we only take the top-K channels in the temporal axis (*i.e.*, the top-k maximum bins). Also, $T(\cdot)$ is a threshold function that masks out bins with number of photons less than a threshold, *i.e.*, reject bins that receive noise instead of a light signal. Once we have obtained the $\mathbf{N}^*_{\mathbf{p}}[R_H, R_V, K]$, we can then transform it to K point clouds as each valid bin (non-rejected) can be back-projected to a point in 3D space. As we use a threshold to mask out invalid points, the number of valid points will be fewer when K is higher, *i.e.*, the 1st strongest point cloud has more points than the 2nd strongest point cloud and so on.

After getting the multi-echo point cloud, we can simulate the reflectance of each point by normalizing the number of points inside the bins, because we assume that each LiDAR sensor transmitter emits same number of photons. The reflectance values of the multi-echo point cloud can be rearranged into the 'LiDAR Image' space $\mathbf{I}_{\mathsf{reflectance}}[R_V, R_H, K]$.

We summarize our multi-echo SPAD-LiDAR simulation process in Algorithm 1 and in Figure 3. In terms of the implementation details, we use our all five camera viewpoints and create a 360° FoV of the multi-echo LiDAR point cloud. We use 10240 numbers of time bins to voxelize 1000 meters of depth range in each view. When we sample the LiDAR sensor array in the polar coordinate, each view has a 35° vertical FoV and 140° horizontal FoV. For example, for frontal view, we sample in $[-17^\circ, 18^\circ]$ vertical FoV with 0.2° of resolution, and $[-60^\circ, 60^\circ]$ horizontal FoV with 0.1° of resolution. To simulate a relatively large spatial coverage (fill factor), we define a $[5, 5]$ patch in image coordinate centered around the projected position as its neighborhood.

2.4 Radar

Similar to the LiDAR data, our Radar data is also simulated via ray-casting. We use the same FoV and sensing range as the high-density long-range LiDAR. The primary difference lies in that the returns of each Radar point contains the velocity of the point in addition to the location information. Such point velocity information can be useful to perception systems and, if used properly, can help prevent collision. To increase realism of the Radar data, we employ the same mechanisms applied to LiDAR points: (1) we randomly drop a small portion of Radar points based on their distance values, *i.e.*, the further the point is, the higher probability it has to be dropped; (2) we randomly perturb a small portion of points along the direction of the ray, creating noisy distance measurements.

2.5 Depth camera

As briefly mentioned in the main paper Sec. 3.1, our sensor suite also contains depth cameras. Here we provide visualization of the depth images. As shown in Figure 4, we encode the depth value in a grayscale image where a larger pixel value (more white) indicates a larger distance.

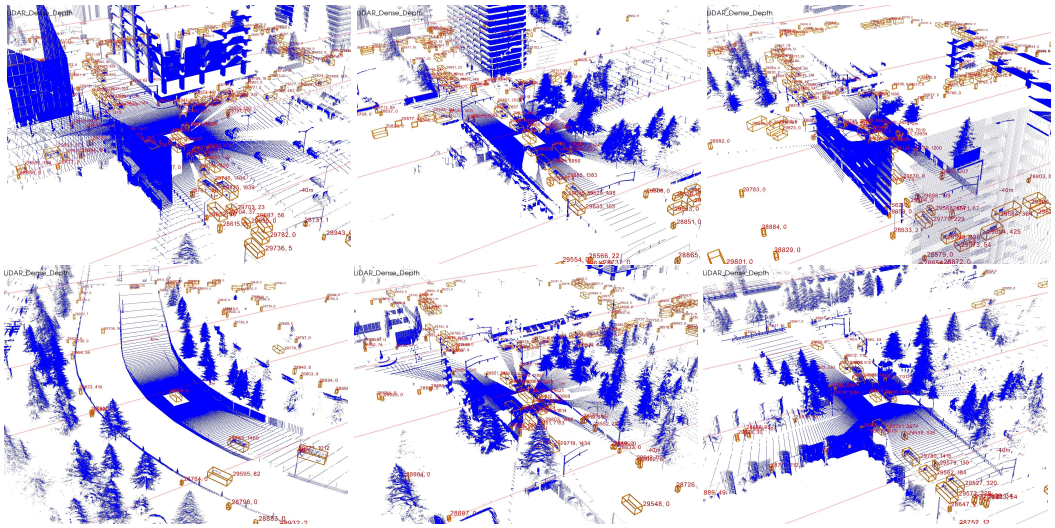

Figure 5: **3D bounding box annotation on depth point clouds.** We show two numbers in the visualization for each agent: (1) ID number (2) number of points inside the box.

3 Diverse annotations

3.1 3D bounding box annotation

In the main paper Sec. 3.2, we have shown visualization of the 2D-3D bounding box annotation in the image. Here, we provide additional visualization of the 3D bounding box annotation in the point cloud in Figure 5. We draw boxes on depth point clouds in the figure but the same 3D box annotation can be drawn on other point clouds as well. As shown in the figure, each bounding box has a unique ID number, which we can be used for 3D object detection, 3D multi-object tracking and trajectory forecasting. Note that our bounding box annotation includes many hard-to-detect objects (*e.g.*, heavily occluded or faraway) which have very few (*e.g.*, 10) points inside the box. Our raw annotation even includes objects that are fully occluded by building with 0 LiDAR point. As fully occluded objects are nearly impossible to detect from a single frame, we use a similar strategy as KITTI by ignoring objects with less than 5 points during 3D object detection evaluation. In other words, missing detecting these objects will not be counted as a false negative while correctly detecting these objects will not be counted as a false positive.

3.2 Fine-grained object class

In addition to high-level object class labels such as car, pedestrian, cyclist as defined in existing datasets, we also provide low-level vehicle model class labels such as Audi A2, Toyota Prius, Tesla Model 3 and Mercedes Benz Coupe. This annotation can be useful to train models for fine-grained vehicle model classification.

3.3 Map

As we use the pre-built Carla city to generate data, the map is known and created in advance. We can map each agent to a location in the map as we have the GPS data for each agent. As a result, our dataset can be used to develop methods for localization, visual odometry, and map-guided trajectory forecasting. For visualization purpose, we have included the map thumbnails in Fig. 6.

4 High environmental variations

4.1 Adverse weather and lighting conditions

Adverse weather and lighting can affect the accuracy of the perception systems. Therefore, it is important to include data under different weather and lighting conditions. In the main paper Sec. 3.3, we briefly mention that our dataset provides data collected with different weathers and lighting. Here, we provide visualization and additional details about how we generate data with different conditions.

As shown in Figure 7, we provide data collected under different weathers such as sunny, cloudy, foggy and rainy. Also, we can simulate the sun rising and falling process in Carla so that we collect data in morning, noon, afternoon, dusk and at night. Among our collected 100 scenes, we have 70

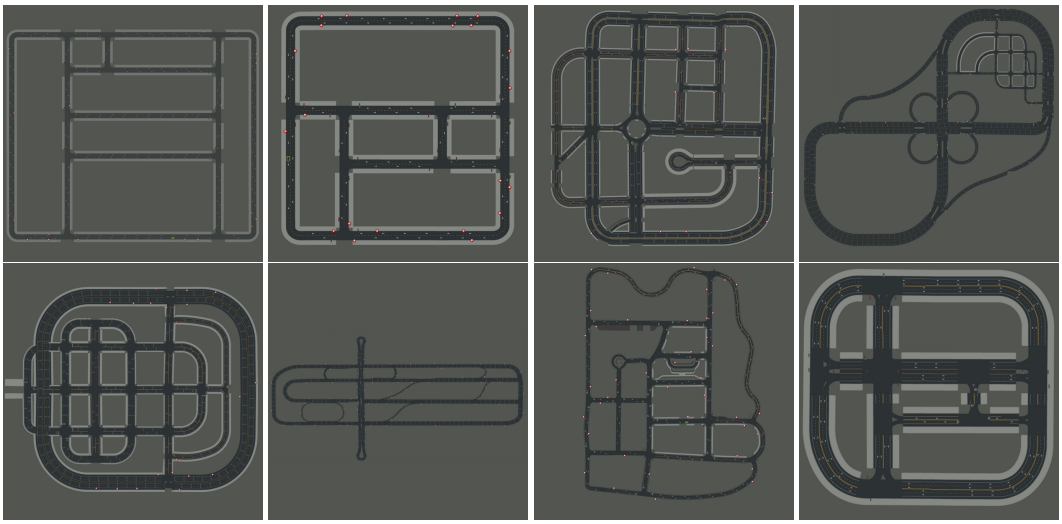

Figure 6: **Map thumbnails.** We visualize the map of eight cities where our data is generated.

Figure 7: Diverse weather and lighting conditions in our collected dataset.

scenes with fixed weather and lighting in typical conditions (cloudy, sunny, morning, noon, afternoon, dusk). For the rest of 30 scenes, we collect data under more challenging situations, *i.e.*, dynamic weather and lighting conditions with rainy, foggy and night included. For example, we control the sun rising from the east and falling in the west, and weather changing from cloudy to rainy.

4.2 Background variations

As stated in the main paper Table 1, our dataset is collected in 8 cities, which covers both urban and suburban areas. As shown in the Figure 5 and 7, our dataset provides data with various backgrounds such as skyscrapers in the downtown and 2-story houses in suburbs, 8-lanes highway and 2-lane suburban road. We believe the diversity of backgrounds in our data is useful for learning robust perception systems that can work in different scene backgrounds.

4.3 Agent behavior customization

As mentioned in the main paper Sec. 3.3, our dataset provides rare driving situations such as accidents, vehicles that run over the light and speed over the limit. The reason we can achieve these rare scenarios is due to the flexible control of every agent provided by the Carla simulator. Specifically, we customize the behaviors of agents for vehicles and people in a very different way. For every vehicle agent, we define a speed range with respect to the speed limit. For example, we can set a vehicle with a minimum of 50% and a maximum of 80% speed with respect to the speed limit of the road, resulting a conservative driver that often drives slowly with a speed less than the speed limit. Also, we can set a vehicle with a maximum of 130% speed with respect to the speed limit,

which will result in an aggressive driver that over-speeds sometimes. In addition to speed control, for every vehicle agent, we also set a different probability of ignoring the traffic light, and ignoring other agents. For example, we can set a vehicle that 100% ignores the traffic light and 50% ignores other agents, which results in a dangerous driver that always runs over the red light and might cause accidents by hitting a pedestrian crossing the road. Additionally, we set for each vehicle agent how frequent to change the lane and how far each vehicle is preferred to be away from other agents (*i.e.*, safe distance) in order to avoid collision.

In addition to controlling vehicles, we also customize the behaviors of pedestrian agents in our dataset. For example, we set different intended speeds for each pedestrian. As a result, our dataset has data with people having a diverse distribution of speed, including pedestrian walking, jogging and running. Also, we set each pedestrian a faraway target destination in the city and also how frequent each pedestrian can cross the road when approaching the destination. With 100% probability of crossing the road, the pedestrians can move towards the destination with a faster route while more probable to cause accidents (*e.g.*, hit by a car). To collect data with different difficulty levels, we generate more aggressive agents in 30 out of 100 scenes with more rare driving cases, and leaving the rest of 70 scenes with most agents behaving normally and conservatively similar to public real-world datasets such as KITTI and nuScenes. We believe that the 30 scenes in our dataset with more aggressive and unpredictable behaviors can serve as a hard evaluation set for tasks requiring predicting agent behaviors such as trajectory forecasting.

5 Justification to the dataset size

As we use the simulator for data generation, we can generate unlimited number of frames in theory and dwarf real datasets. However, dealing with a large-scale dataset is non-trivial. After optimizing our code to increase the speed, our current code for generating and post-processing the dataset with 100k frames still requires a few weeks of time. Also, storage is another constraint. Our current dataset, including all sensor data and annotations, requires about 3Tb of storage. With more frames to be generated, the required storage will be bigger which can cause inconvenience for users. To ease the use, we only generate 100k frames for now. Also, we suggest users only downloading the sensor data and annotation they need for their models and corresponding tasks.

6 Experiments

6.1 Data split

We split our dataset into train, validation and test splits, each with 40, 30, 30 scenes of the total 100 scenes. The split is based on the environmental variations including diversity of background, weather/lighting and agent behaviors. The goal is to make sure that data in each split still has diverse environmental variations. We will release the sensor data for all three splits while only release the ground truth for train and validation splits so that the testing split can be reserved for fair comparison using our evaluation server.

6.2 Object detection evaluation protocol

Difficulty level. We use three difficulty levels (easy, moderate and hard) following KITTI. For 2D detection, we define the difficulty level based on the ground truth 2D box height of 40/25/15 pixels and 2D occlusion/truncation percentage of 30%/50%/70%. If a ground truth object satisfies any of requirement to be in a more difficult level, then we define it as in that more difficult level. For example, if a ground truth object has a occlusion level of 80%, though its box height is 20 pixels, we define this object to be in the hard level instead of the moderate level. Similarly, for 3D detection, we define the difficulty level based on the distance of 40/80/120 meters. For both 2D and 3D object detection, we define an ignored object set which includes objects with less than 5 points inside their 3D boxes. Objects in this ignored set are not required to be detected by detectors as these objects are nearly invisible in all sensors such as camera and LiDAR.

Matching criteria. When deciding if a detected object is a true positive, *i.e.*, corresponding to a ground truth object, we use a 2D IoU (Intersection of Union) threshold of 0.7 for car and 0.5 for pedestrian and cyclist in 2D object detection evaluation. For 3D detection evaluation, we use a 3D IoU threshold of 0.7 for car, and 0.5 for pedestrian and cyclist.

6.3 Implementation details for 3D detection

As mentioned in the main paper Sec. 4.2, we have increased the input point cloud range from 0-70m in KITTI to 120m in 360° degrees (*i.e.*, a circle with a diameter of 240 meters) for 3D detectors to enable perception at a larger range, nearly quadrupling the detection range in KITTI. We choose to increase the detection range to 120 meters (not even larger *e.g.*, 1000 meters) because current state-of-the-art methods are not memory efficient. As a result, further enlarging the detection range can easily reach GPU memory limit, *e.g.*, detection up to 1000 meters requires GPU memory larger than 1Tb to process the point clouds for voxel-based methods without sacrificing the voxelization resolution too much. Future work in memory-efficient 3D detection is needed to fully leverage long-range high-density point clouds to perform perception at a larger range.

In addition to 120 meters of horizontal detection range, we use a detection range of $[-5, 2]$ meters in the height direction, which covers nearly all agents even including those on a slope with a negative height, *e.g.*, -4m. For all baselines, we use anchors with width, length and height of (1.6, 3.9, 1.6)m, (0.8, 0.8, 1.7)m and (0.6, 1.8, 1.7)m for cars, pedestrians and cyclists, respectively. Also, for voxel-based methods (SECOND and PointPillars), we slightly decrease the voxelization resolution to reduce memory requirement as otherwise 120 meters of input point clouds cannot fit into GPU memory. Specifically, we use a voxel size of $0.1m \times 0.15m \times 0.1m$ (in length, height, width directions) for training on cars, and $0.125m \times 0.3m \times 0.125m$ for training on pedestrians and cyclists.

For experiments in the main paper Table 5, all 3D detection baselines are trained from scratch using only data from the AIODrive train split. We train every baseline for 7 epochs for a fair comparison using 4× NVIDIA GeForce RTX 2080. For experiments in the main paper Table 6 using SPAD-LiDAR data (the last row), we use the top-2 strongest point cloud returns as input to PointRCNN instead of the raw 3D tensor. As PointRCNN cannot explicitly leverage the relationship between different SPAD-LiDAR point cloud returns (or echos), we simply concatenate the top-2 point clouds together to form a single point cloud as input to PointRCNN.

7 Data hosting and maintenance

As directed by our data download links (http://www.aiodrive.org/download.html), all of our data are hosted on Dropbox with shareable links so that everyone on the internet can download. As our dataset is about 3 Tb, we use the Dropbox professional plan. With affordable price ($200 per year with unlimited download) on Dropbox, it should be easy for us to keep data download link alive for at least a few years until the dataset is no longer useful to the community.

8 Dataset license

Our AIODrive dataset and associated code are released under Creative Commons Attribution-ShareAlike 4.0 International Public License. So everyone is free to use for both commercial and research purpose.

References

[1] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds. *CVPR*, 2019.

[2] Velodyne Lidar. High Definition Real-Time 3D Lidar. `https://velodynelidar.com/products/hdl-64e/`.

[3] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-Aware and Part-Aggregation Network. *CVPR*, 2020.

[4] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely Embedded Convolutional Detection. *Sensors*, 2018.