applied sciences

MDPI

Article

CarFree: Hassle-Free Object Detection Dataset Generation Using Carla Autonomous Driving Simulator

Jaesung Jang [1], Hyeongyu Lee [1] and Jong-Chan Kim [2,*]

[1] Graduate School of Automotive Engineering, Kookmin University, Seoul 02707, Korea; jangjs1221@kookmin.ac.kr (J.J.); a2020011@kookmin.ac.kr (H.L.)
[2] Department of Automobile and IT Convergence, Kookmin University, Seoul 02707, Korea
* Correspondence: jongchank@kookmin.ac.kr

Abstract: For safe autonomous driving, deep neural network (DNN)-based perception systems play essential roles, where a vast amount of driving images should be manually collected and labeled with ground truth (GT) for training and validation purposes. After observing the manual GT generation's high cost and unavoidable human errors, this study presents an open-source automatic GT generation tool, CarFree, based on the Carla autonomous driving simulator. By that, we aim to democratize the daunting task of (in particular) object detection dataset generation, which was only possible by big companies or institutes due to its high cost. CarFree comprises (i) a data extraction client that automatically collects relevant information from the Carla simulator's server and (ii) a post-processing software that produces precise 2D bounding boxes of vehicles and pedestrians on the gathered driving images. Our evaluation results show that CarFree can generate a considerable amount of realistic driving images along with their GTs in a reasonable time. Moreover, using the synthesized training images with artificially made unusual weather and lighting conditions, which are difficult to obtain in real-world driving scenarios, CarFree significantly improves the object detection accuracy in the real world, particularly in the case of harsh environments. With CarFree, we expect its users to generate a variety of object detection datasets in hassle-free ways.

Keywords: object detection; ground truth generation; Carla autonomous driving simulator

Citation: Jang, J.; Lee, H.; Kim, J.-C. CarFree: Hassle-Free Object Detection Dataset Generation Using Carla Autonomous Driving Simulator. *Appl. Sci.* **2022**, *12*, 281. https://doi.org/10.3390/app12010281

Academic Editors: Javier Alonso Ruiz, Kristina Yordanova and Emma Tonkin

Received: 12 November 2021
Accepted: 23 December 2021
Published: 28 December 2021

1. Introduction

For autonomous driving, developing camera-based perception systems requires a tremendous amount of training images, usually collected and labeled by human labor that is costly and error-prone. To make things worse, gathering such a vast amount of real-world driving images with enough diversity is challenging because we cannot artificially make unusual corner cases or peculiar weather and lighting conditions. Thus, few companies and institutes with significant budgets and resources can afford it.

In recent times, synthesized datasets from 3D game engines are gaining wide acceptance as a viable solution [1–4]. This new approach has been advocated with compelling results when training deep neural networks (DNNs) for perception systems [5,6]. Not surprisingly, big industry players [7,8] are also known to use similar approaches in their proprietary products. Even with the great potential of such synthesized datasets, to the best of our knowledge, there is no publicly available or open-source tool for automatically generating such datasets. Meanwhile, there are commercial dataset generation services [9,10] possibly with their proprietary dataset synthesis pipelines, indicating the existing needs for such tools.

Motivated by these observations, as an initial effort towards democratizing the daunting task of dataset generation for perception systems, this study presents an open-source image synthesis and automatic ground truth (GT) generation software, particularly for camera-based vehicle and pedestrian detection. There need various datasets such as 2D object detection, 3D object detection, and semantic segmentation datasets. Among them,

this study focuses on synthesizing 2D object detection datasets. Figure 1 shows our synthesized example images with four different driving scenarios (by the rows) under varying weather conditions (by the columns). On each image, the red bounding boxes (vehicles) and green bounding boxes (pedestrians), as well as the image itself, were automatically generated by our tool, which we named *CarFree*.

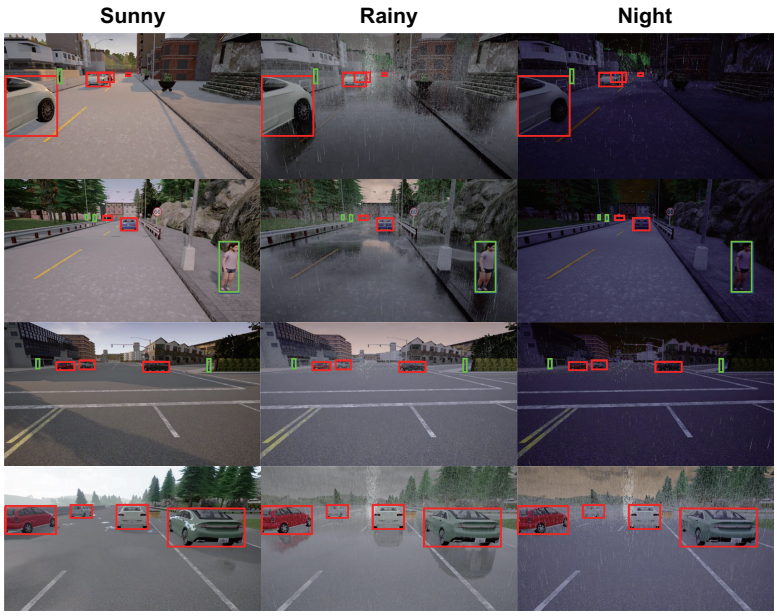


Figure 1. Selected images synthesized by CarFree with automatically generated 2D object detection ground truths (GTs), represented by the red and green bounding boxes for vehicles and pedestrians, respectively. The three columns represent different weather and lighting conditions for the four driving scenes by the rows.

CarFree is based on the Carla autonomous driving simulator [11], which was chosen after carefully evaluating recent autonomous driving simulators [11–14] to find the best fit with our research objective. More specifically, we have chosen Carla due to its wide acceptance in research communities and the fast development progress, as well as its well-defined client-server architecture. In its action, the Carla server simulates a pre-defined virtual world (so-called a *town*), where the clients interact with the server with its Python application programming interface (API). We can control the weather, the ego vehicle, the surrounding vehicles, and the pedestrians in the town through the provided API. Moreover, we can extract the ego vehicle's sensor data (e.g., forward-facing camera) and the physical properties (e.g., location, pose, and speed) of surrounding vehicles and pedestrians (henceforth referred to as *objects*).

However, even with the abundant information available from the Carla server, unfortunately, Carla does not directly provide GTs (i.e., bounding boxes) for 2D object detection. Thus, we had no choice but to manually generate bounding boxes at first, wasting a considerable amount of time. Nonetheless, after carefully investigating the data available from the Carla server, we concluded that exact 2D bounding boxes can be derived from the raw data.

For that purpose, we found that the following pieces of information from the Carla server are useful:

- **3D bounding boxes** that represent the location and the pose of every object in the 3D coordinate;
- **Camera images** that are 2D RGB data from the virtual forward-facing camera installed on the ego vehicle;

- **Semantic segmentation images** that are pixel-wise classifications by different color codes for each corresponding camera image (e.g., blue for vehicles and red for pedestrians).

Note that even with the semantic segmentation images, since they treat multiple objects of the same class as a single entity, we cannot individually recognize overlapped objects. Further, we need to handle the difference between the 3D coordinate and the 2D image coordinate, considering the effect of occlusions by closer structures. Thus, a new GT generation method needs to be developed to handle the above challenges.

Our automatic dataset generation tool, CarFree, comprises two parts: (i) a data extraction part and (ii) a GT generation part. The first part is implemented as Carla clients in Python that deploys an ego vehicle in the town virtually made in the Carla server, where the ego vehicle can be driven either manually or autonomously. While driving, it can periodically gather raw data from the Carla server; the user can also intervene to capture a scene once spotting a scarce corner case. Then our GT generation algorithm in the second part processes the raw data to find precise 2D bounding boxes for every object.

CarFree is evaluated in terms of the following:

- **Scalability.** How many images can be processed while finding their GTs in a given time?
- **Precision.** How precise are the generated GTs in terms of locating bounding boxes?
- **Training performance.** How effective is the generated dataset for training DNNs in terms of their object detection performance?

Regarding the *scalability*, our (single-thread) GT generation process can handle about five images (assuming six objects in each) per second on average, which is about 75 times faster than experienced human workers. Further, the process is parallelizable using multi-core processors due to the inherent data parallelism. For the *precision*, we visually examined our 5000 generated images by human eyes, finding that every recognizable object is successfully identified inside appropriate bounding boxes. Besides, our generated GTs guarantee pixel-level precision for locating bounding boxes. To evaluate the *training performance* of the synthesized datasets, we need a canonical object detection network. Among many state-of-the-art networks, we employ YOLO v3 [15] based on the Darknet framework [16]. Our evaluation results show that the generated dataset provides compelling training results, in particular when used along with real-world driving images. Furthermore, it significantly improves the object detection accuracy for perceiving driving scenes in unusually harsh environments such as rain, storm, and fog. We claim that the improvements are achieved by our synthesized training images that exhibit a great deal of diversity in terms of weather and lighting conditions provided by CarFree.

The major contributions of this study can be summarized as follows:

- To the best of our knowledge, this study is one of the first works to propose public domain, open-source tools for automatically synthesizing 2D object detection datasets for autonomous driving.
- Moreover, this study shows that by using the synthesized training dataset with diverse weather and lighting conditions, the accuracy for the real-world object detection can be significantly improved.

The rest of this paper is organized as follows: Section 2 provides related work. Section 3 provides the background information. Section 4 explains the overall architecture of our solution approach. Section 5 presents our bounding box generation algorithm. Then our approach is evaluated in Section 6. Finally, Section 7 presents conclusions for this study.

2. Related Work

Object detectors. Due to the recent progress in the deep learning-based models, most modern object detection systems are based on DNNs [15,17–21]. Among them, two-stage object detectors [17,18] are usually better in their accuracy, whereas single-stage object detectors [15,19–21] are superior in the real-time performance. To train and deploy

object detection models, there are object detection frameworks such as Darknet [16] and Detectron2 [22]. In this study, we use the Darknet framework along with the YOLO object detector for both the training and the inference. Our choice is based on the fact that they are commonly used in recent autonomous driving systems [23–25].

Datasets. There are famous datasets such as ImageNet [26], PASCAL VOC [27], and MS COCO [28] for training and evaluating general-purpose object detectors. However, since they include objects that are irrelevant for understanding driving scenes [29], specialized datasets for autonomous driving such as KITTI [30], Cityscape [31], Mapillary Vistas [32], and BDD100K [33] are more often used in the research community. In recent times, datasets for more complex perception tasks with multimodal sensors, such as Argoverse [34], ApolloScape [35], NuScenes [36], Waymo Open Dataset [37], and Lyft Level 5 AV Dataset [38], are being released.

Synthesized Datasets. Due to the excessive efforts and costs for gathering real-world driving images, they are usually produced by big companies and institutes. Thus, generating synthetic driving scenes from 3D game engines like Unity [39] and Unreal Engine [40] is being considered as a more cost-effective alternative. Virtual KITTI [2], SYNTHIA [1], and Playing for Data [41] are some of the first such attempts. Synthetic images are especially valuable when dangerous driving scenarios or unusual whether conditions are needed, because we have the power to freely control weather in the virtual world. Even with the synthetic images, however, human labors are still required while labeling GTs for the generated images, which takes excessive time and cost. For example, it reportedly takes 88 s on average for an experienced human worker to draw and verify a single bounding box [42], which is much too burdensome.

Labeling tools. There are various tools for the manual GT generation, which assist human workers to annotate the location of objects. For that, the tools provide user interfaces for drawing bounding boxes, polygons, as well as pixel-level classifications sometimes. There are well-known tools available such as CVAT [43], VoTT [44], COCO Annotator [45], VIA [46], YOLO mark [47], and LabelMe [48]. Besides, recent studies try to automatically propose GT candidates by preprocessing the images with their pre-trained object detection models [49–51]. This semi-automation helps human workers draw bounding boxes faster than they draw out of nothing but the images. Even with the above user interfaces and semi-automation methods, they are still far from full automation and impose too much burden due to the inordinate amount of images to be processed.

Autonomous driving simulators. Game engine-based autonomous driving simulators have recently gained wide acceptance in the research community, as well as in the industry. In particular, open-source simulators such as AirSim [12], LGSVL Simulator [13], and Carla [11] are widely used in the research community. Among them, our study uses the Unreal Engine [40]-based Carla autonomous driving simulator, which was chosen mainly due to its rich set of Python APIs and the well-separated client-server architecture. Note that Carla is widely used for validating decision logics [52–54] and also for training agents using reinforcement learning techniques [55–57]. In some cases, Carla is used for evaluating camera-based perception systems [58] because it can generate photorealistic driving scenes.

3. Background

3.1. Object Detection for Autonomous Driving

This study targets the 2D object detection task that plays a crucial role in autonomous driving systems. Since dataset generations require enormous human labor, the research communities cannot help relying on public datasets [30,31,33] for the evaluation of their DNN models. However, public datasets are inherently limited in quantity and diversity compared with proprietary datasets owned by big companies. For example, Tesla claims that they use 6 billion object labels for training their neural networks [59]. Gathering more data and corner cases is becoming more critical than inventing new DNN architectures these days.

With the above motivations, our goal is to develop a tool that can generate object detection datasets practically at no cost by automatically synthesizing driving scenes and extracting GTs from them. For the evaluation, we looked into various DNN models [15,17–21] and object detection frameworks [16,22]. Among them, this study uses the YOLO v3 DNN with the Darknet framework. YOLO is widely used in autonomous driving systems due to its real-time performance as well as its state-of-the-art detection performance. Darknet is chosen because it is written entirely with the C language and the CUDA language, making it highly portable across different hardware platforms. However, we believe that our tool can be helpful to any object detection model and framework.

3.2. Carla Autonomous Driving Simulator

In Carla, there are three important concepts: (i) *world*, (ii) *actors*, and (iii) *blueprints*. During the simulation, a Carla client first creates its world in the server with a pre-defined map. In such a simulated world, actors are spawned that can be any object playing a certain role in the simulation, usually moving around the town (e.g., vehicles, pedestrians, and sensors). Actors are spawned out of blueprints, which is a 3D model with animation effects. Carla provides a set of ready-to-use baseline blueprints. For example, there are blueprints for various vehicle models, pedestrians with varying appearances (e.g., male, female, and child), and various sensors (e.g., camera, lidar, and radar). After creating a world with necessary actors, the simulation loop begins, where the client can control the weather and lighting conditions and move the ego vehicle, surrounding vehicles, and pedestrians. Besides, at each simulation step, the client can extract various information from the server. In particular, CarFree collects the following data that are necessary for generating 2D bounding boxes:

- **3D bounding boxes and classes:** Assuming n actors operating in the town, n corresponding 3D bounding boxes are extracted. A 3D bounding box is denoted by its eight corner points in the set of 3D bounding boxes $\{(C_1^1, C_1^2, \cdots, C_1^8), \cdots, (C_n^1, C_n^2, \cdots, C_n^8)\}$, where each point C is represented by (x, y, z) in the 3D coordinate. We can also find each actor's class, denoted by T_i $(1 \leq i \leq n)$, which is either *vehicle* or *pedestrian*.
- **Camera images:** A forward-facing camera is installed on the ego vehicle, from which a 960×540 resolution RGB image can be retrieved. Each image is given by a 960×540 matrix I, where each element $I[x][y]$ represents the color code of the pixel at the location (x, y) in the 2D image coordinate.
- **Semantic segmentation images:** For each RGB image, its corresponding semantic segmentation image is also extracted, given by a 960×540 matrix S, where each element $S[x][y]$ represents the class the object the pixel belongs to by the pre-defined color codes.

Note that we have three different coordinates: (i) the *world coordinate* (3D) where its coordinate origin is the center of the town, (ii) the *vehicle coordinate* (3D) where its coordinate origin is the center of the ego vehicle, and (iii) the *image coordinate* (2D) where its coordinate origin is the left-top corner of an image.

4. Overall Architecture

Figure 2 shows the overall design of CarFree that comprises the following parts: (i) the Carla server in the left, (ii) the data extraction part in the middle, and (iii) the GT generation part in the right.

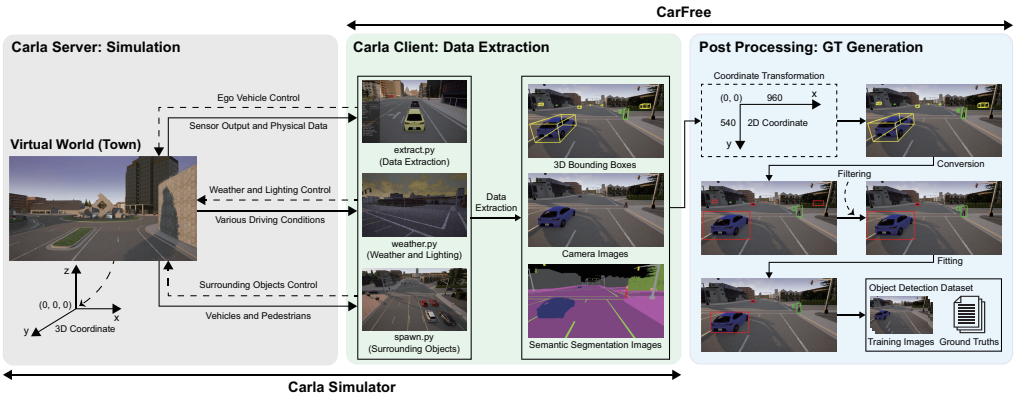


Figure 2. Overall architecture of CarFree.

Regarding the server, we use the unmodified stock version from the Carla repository. Thus, if Carla is already installed in the system, it can be used without any modification. The data extraction part is implemented as three Carla clients as listed in the middle of Figure 2. They control the simulated world in terms of its weather and lighting conditions (weather.py) and the surrounding vehicles and pedestrians (spawn.py). Then it extracts the data listed in Section 3.2 (extract.py). Since they are implemented as Carla clients, the data extraction stage works in real time, by the time progress in the Carla server. In contrast, the GT generation stage runs as a separate process that works offline. It calculates the 2D bounding boxes of vehicles and pedestrians from the extracted data as in the following steps:

- First, the eight points of each 3D bounding box are projected to the 2D image coordinate of the forward-facing camera (Section 5.1).
- Second, for the projected eight points, their minimum area bounding box that barely encloses them is derived (Section 5.2).
- Third, occluded objects are removed because, for example, objects behind a wall are still present at this stage (Section 5.3).
- Fourth, each bounding box is tightly fitted to the object by removing the gap between the initial bounding box and the actual object boundary (Section 5.4).

The resulting bounding boxes should be recorded in an appropriate file format. For that, there is no standard format, but we have several candidates for that purpose. For example, the Darknet object detection framework [16] employs a plain text file format with object class numbers and 2D bounding boxes represented by its center point (x, y) along with its height (h) and width (w). Other frameworks such as Detectron2 use a JSON (JavaScript Object Notation)-based file format with slightly different representation methods. This study follows the Darknet's convention. However, because the conversion between different formats is straightforward, we can easily apply our method to other frameworks.

5. Automatic Ground Truth Generation

This section explains the GT generation part (i.e., the rightmost part in Figure 2) in more detail, which comprises the following four steps: (i) coordinate transformation, (ii) conversion to 2D bounding boxes, (iii) filtering out occluded objects, and (iv) fitting to object boundary. Assuming multiple objects in a camera image, CarFree takes care of each object one by one. Thus, this section simply explains how we extract the bounding box of a single object. CarFree iterates the following process for every object in the town, finding every visible object inside the camera's viewpoint, which is straightforward.

5.1. Coordinate Transformation

As mentioned earlier, three different coordinate systems (i.e., the world coordinate, the vehicle coordinate, and the image coordinate) coexist in Carla. Carla provides the 3D

bounding boxes of every object in the town in the vehicle coordinate. Besides, the positions of the ego vehicle and the camera are provided in the world coordinate. Then our problem here is to transform the eight points of a 3D bounding box in the vehicle coordinate into their projection points in the 2D image coordinate of the forward-facing camera.

For that, each point (x, y, z) in the 3D vehicle coordinate is transformed into the 2D image coordinate through the 3D world coordinate and a temporary camera coordinate. Since we know the ego vehicle's center point in the world coordinate, any given point in the vehicle coordinate can be transformed into the world coordinate using a transformation matrix. Then, for the perspective transformation to the camera's viewpoint, the camera's position is also incorporated, making a temporary 3D camera coordinate with its z-axis parallel to the camera's viewing direction. Now, for each point in this 3D camera coordinate, the camera's field of view (FoV) and its image resolution are considered for the perspective transformation. Any given point in the 3D vehicle coordinate can be transformed to its corresponding point in the 2D image coordinate by the above steps.

5.2. Conversion to 2D Bounding Boxes

The next step is to make an initial proposal for the 2D bounding box out of the eight projection points, which is depicted in Figure 3a. In the leftmost part, the eight corner points of the yellow cuboid represent a projected 3D bounding box in the 2D image coordinate. For each cuboid, its minimum area bounding box is found, which is depicted as a red rectangle in the rightmost part of the figure.

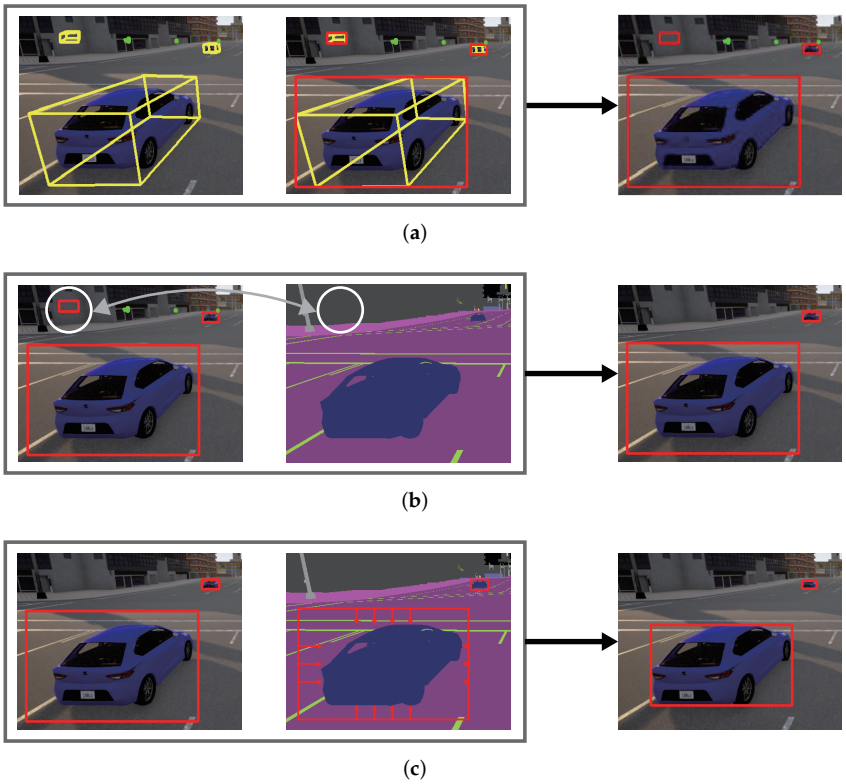


Figure 3. Post-processing for the GT generation. (**a**) Conversion to 2D bounding boxes. (**b**) Filtering out occluded objects. (**c**) Fitting to object boundary.

Algorithm 1 shows the procedure how the eight 2D points $\{(x_1, y_1), \cdots, (x_8, y_8)\}$ representing a projected 3D bounding box can be converted into a 2D bounding box represented by $\{(\hat{x}_1, \hat{y}_1), \cdots, (\hat{x}_4, \hat{y}_4)\}$. Although now we have an initial proposal, there are two remaining issues. The first problem is that invisible objects due to occlusions are present. For example, in Figure 3a, the object for the 3D bounding box in the left top corner is behind a building, which should be removed. The second issue is the considerable gap

between the bounding box and the object boundary. It is undesirable because imprecise bounding boxes can lead to significant detection errors of the trained model.

Algorithm 1: Conversion to 2D bounding boxes

Require: $\{(x_1, y_1), \cdots, (x_8, y_8)\}$
Ensure: $\{(\hat{x}_1, \hat{y}_1), \cdots, (\hat{x}_4, \hat{y}_4)\}$
$x_{min} \leftarrow \min\{x_1, \cdots, x_8\}$
$x_{max} \leftarrow \max\{x_1, \cdots, x_8\}$
$y_{min} \leftarrow \min\{y_1, \cdots, y_8\}$
$y_{max} \leftarrow \max\{y_1, \cdots, y_8\}$
return $\{(x_{min}, y_{min}), (x_{max}, y_{min}),$
$(x_{min}, y_{max}), (x_{max}, y_{max})\}$

5.3. Filtering Out Occluded Objects

This section introduces our idea for identifying occluded objects, which is depicted in Figure 3b. For that, we make use of the semantic segmentation image, which is also available from the Carla server. A semantic segmentation image provides pixel-wise classifications where each pixel represents an object's class by a color code. For example, in the middle of Figure 3b, the pixels that belong to vehicles are represented by blue color. It also indicates that the proposed bounding box in the left top corner is actually not a visible object since a building occludes it. Thus, it can be safely ignored.

Algorithm 2 shows the procedure of how we can check the existence of a valid and visible object inside a given 2D bounding box. The algorithm takes a 2D bounding box, its corresponding object class, and a semantic segmentation image. Then, if the center pixel of the bounding box is identified as belonging to the given object class by checking the semantic segmentation image, it is decided as visible. Note that our algorithm may fail to identify a partially occluded object, which is a known limitation of our method. For that, please understand that our primary aim is not to discover all the fully and partially visible objects. Instead, we aim to produce as many training images as possible from the simulator. For finding partially occluded objects, Algorithm 2 can be easily extended by sampling multiple locations in addition to the center point.

Algorithm 2: Check the visibility of an object

Require: $\{(x_1, y_1), \cdots, (x_4, y_4)\}$
Require: T: Target object class
Require: S: Semantic segmentation image
Ensure: True when the object is visible, False otherwise
$x \leftarrow (x_1 + x_2 + x_3 + x_4)/4$
$y \leftarrow (y_1 + y_2 + y_3 + y_4)/4$
if $S[x][y] == T$ **then**
    return True
else
    return False

Algorithm 3 shows the fitting procedure, where the original bounding box $\{(x_1, y_1), \cdots, (x_4, y_4)\}$ is given as an input and its output is a fitted bounding box $\{(\hat{x}_1, \hat{y}_1), \cdots, (\hat{x}_4, \hat{y}_4)\}$.

Algorithm 3: Fitting to object boundary

Require: $\{(x_1, y_1), \cdots, (x_4, y_4)\}$
Require: T: Target object class
Require: S: Semantic segmentation image
Ensure: $\{(\hat{x}_1, \hat{y}_1), \cdots, (\hat{x}_4, \hat{y}_4)\}$
$x_{min} \leftarrow \min\{x_1, x_2, x_3, x_4\}$
$x_{max} \leftarrow \max\{x_1, x_2, x_3, x_4\}$
$y_{min} \leftarrow \min\{y_1, y_2, y_3, y_4\}$
$y_{max} \leftarrow \max\{y_1, y_2, y_3, y_4\}$
while True **do**
  if T is not in $S[x_{min}]$ **then**
    $x_{min} = x_{min} + 1$
  if T is not in $S[x_{max}]$ **then**
    $x_{max} = x_{max} - 1$
  if T is not in $S[y_{min}]$ **then**
    $y_{min} = y_{min} + 1$
  if T is not in $S[y_{max}]$ **then**
    $y_{max} = y_{max} - 1$
  if no more progress **then**
    break
return $\{(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{max})\}$

5.4. Fitting to Object Boundary

In the leftmost part of Figure 3c, notice the considerable gap between the initially proposed bounding box and the boundary of the actual vehicle. To make it more precise, we need to fit the bounding box to the exact area occupying the object. For that, the semantic segmentation image is once again utilized. The middle part of Figure 3c illustrates the fitting process. It begins with the initial bounding box, and the four sides of it gradually moves toward their opposite directions until all of them meet at least one pixel that belongs to the target object. After that, the resulting bounding box becomes tightly fitted to the object, as shown in the rightmost part of Figure 3c.

6. Experiments

6.1. Implementation

For the implementation, CarFree is developed based on Carla 0.9.7, and the source code is available at its GitHub repository (Available at https://github.com/AveesLab/CarFree (accessed on 3 September 2021)). Nonetheless, because CarFree is implemented as separate Carla client programs and Python scripts, it can be used with existing Carla server instances. More specifically, CarFree's data extract part is composed of the following three Python scripts:

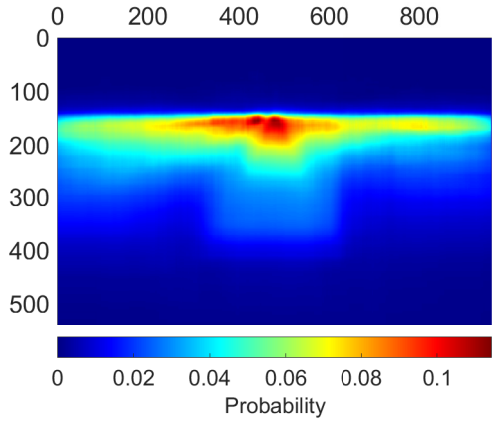
- **spawn.py** for deploying required vehicles and pedestrians in the town,
- **weather.py** for controlling the weather and lighting conditions, and
- **extract.py** for controlling the ego vehicle and extracting data from the Carla server.

CarFree supports two different operation modes. One is a periodic data extraction that continually retrieves data with a period given by a command-line option. The other is an event-based data extraction triggered by a user's keypress, useful when the user accidentally finds a corner case scenario while monitoring the simulation. In either case, the extracted data is written to PNG image files (camera images and semantic segmentation images) and text files (3D bounding boxes) in a designated directory. Then the post-processing GT generation script processes them as explained in Section 5, producing the final GTs.

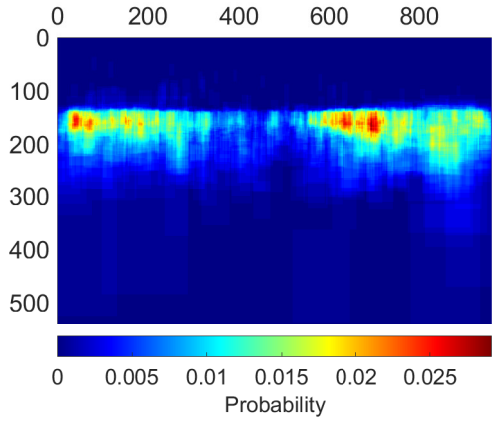
6.2. Evaluation

This section evaluates CarFree in terms of its (i) scalability, (ii) precision, and (iii) training performance. For the evaluation, we synthesized 5000 training images and GTs using CarFree in four different towns (i.e., Town1, Town2, Town3, and Town4 provided by Carla). During each simulation, 50 random vehicles and 30 random pedestrians from the Carla blueprints were deployed, moving around the town autonomously by its autopilot module. An ego vehicle is deployed with a forward-facing camera with a 90 degrees FoV that captures images with a resolution of 960 $\times$ 540. The altitude and azimuth of the Sun rapidly change, simulating various lighting conditions. The weather condition is also controlled by changing Carla's weather parameters (i.e., cloudiness, precipitation, and windiness).

To see where objects appear in the synthesized images, Figure 4 shows the heatmaps along the 2D image coordinate. The heatmaps are made by calculating the pixel-wise intensity of appeared objects. For that, we counted each pixel's covering bounding boxes for vehicles and pedestrians across all the synthesized images. Figure 4a shows that vehicles appear along the horizon with more intensity around the vanishing point. Figure 4b shows that pedestrians are more likely to appear on the left and right sides of the images because they usually walk along the sidewalks. However, some pedestrians randomly cross the road, indicated by the bright points in the center area. The number of objects randomly varies in each image. In summary, 27% of the images have only one object; 36% of the images contain two objects; 20% of the images contain three objects; 18% of the images have four or more objects.



(**a**)



(**b**)

Figure 4. Heatmaps for the generated objects in the 2D image coordinate. (**a**) Vehicles. (**b**) Pedestrians.

Scalability. Figure 5 compares the average performance (i.e., images per second) of the GT generation process of CarFree with experienced and inexperienced human workers. Their performances are evaluated with six different image sets with 150 images each. The images are grouped by the number of objects in each image, from one object to six objects. For the experiment, we employed a volunteer who had no experience with any GT labeling tool. We also employed another volunteer who previously had done lots of GT labeling jobs. The figure shows that CarFree can generate a much larger number of images than human workers do. For example, for the one object case, CarFree's throughput is 1948% compared with the experienced worker and 3491% compared with the inexperienced worker. With more objects in each image, the performance gap generally gets larger. The figure also shows that the number of objects in each image significantly affects the labeling performance of the human labor, whereas the effect is less significant in CarFree. More specifically, by comparing the case of six objects with the case of one object, CarFree's performance dropped from 8.40 (100%) to 4.85 (58%), whereas the experienced worker's performance dropped from 0.43 (100%) to 0.07 (16%), and the inexperienced work's performance dropped from 0.24 (100%) to 0.047 (20%).

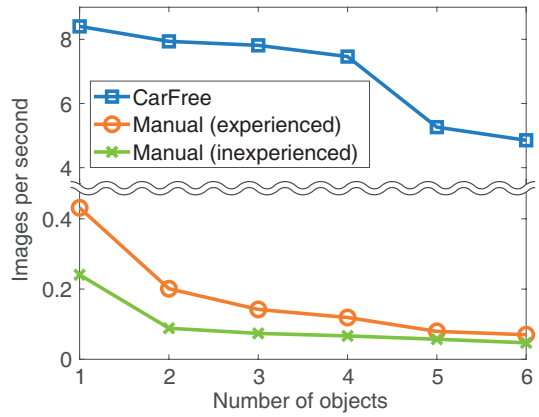


Figure 5. Evaluation for the scalability.

Precision. Throughout the generated training images, we visually examined their precision by human eyes and confirmed that every relevant object is successfully located by our GT generation process. Further, as exemplified by Figure 6, our generated bounding boxes are with pixel-level precision, whereas manual labeling cannot produce such precise bounding boxes. Moreover, note that tiny distant objects are easily ignored by human eyes, whereas CarFree does not miss even such tiny objects.

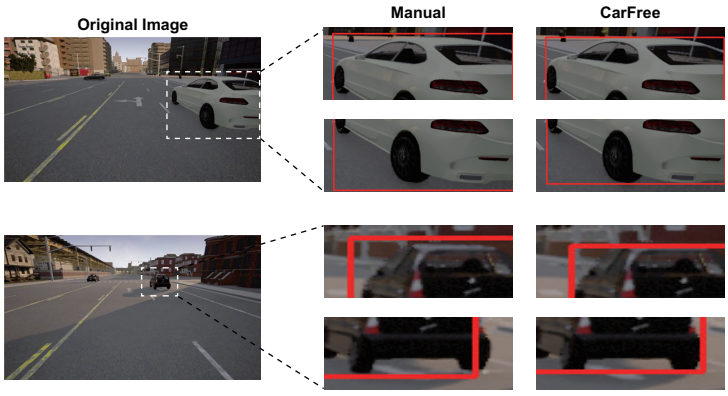


Figure 6. Evaluation for the precision.

Training performance. Although our synthesized images can reinforce the training process, purely using synthesized images is usually not a practical choice. Instead, it

is reported that combining real-world images and synthesized images produces better training results [3]. Based on this practice, besides our synthesized images, we also use the KITTI [30] dataset and the BDD100K [33] dataset, both of which contain real-world driving images. We use 500 randomly chosen KITTI images as the baseline training images. Then we add more and more images to evaluate how the added images perform to enhance the object detection accuracy. The images from the BDD100K dataset are used for evaluation purposes. The roles of these two real-world datasets were decided after carefully comparing the two datasets. The KITTI dataset only contains typical driving images under good weather conditions, whereas the BDD100K dataset provides more diverse images with various weather conditions. Thus, we concluded that the BDD100K images are better suited as the evaluation dataset because they better reflect the real world's diversity. Then our primary focus is to evaluate how our synthesized images can reinforce the baseline training images. For the object detection model, we used the YOLO v3 DNN based on the Darknet framework.

The training begins with the 500 baseline images from the KITTI dataset. Then we use the following three training strategies and evaluate how well they reinforce the training:

- **KITTI only:** In addition to the baseline images, other random KITTI images are incrementally added as well.
- **KITTI+CarFree (random):** In addition to the baseline images, our synthesized images are incrementally added in random.
- **KITTI+CarFree ($\geq$4 objects):** In addition to the baseline images, our synthesized images with at least four objects are incrementally added.

The difference between **KITTI+CarFree (random)** and **KITTI+CarFree ($\geq$4 objects)** is that the latter is likely to provide more complexity and diversity, which are expected to have positive effects on the training performance.

Figure 7a compares the object detection accuracy, i.e., mAP (mean average precision), of the above three training strategies with varying number of training images. The x-axis begins with 500, indicating the number of baseline images. The mAP is measured by the evaluation set with 200 random BDD100K images. As shown in the figure, as we add more and more training images, the performance generally gets better. Moreover, it shows that **KITTI + CarFree (random)** outperforms **KITTI only**, and **KITTI + CarFree ($\geq$4 objects)** performs even better. More specifically, with 2000 training images, **KITTI only**'s mAP is 74.51, whereas **KITTI + CarFree (random)** reaches 77.06 and **KITTI + CarFree ($\geq$4 objects)** reaches 79.08. The results indicate that the diversity, which is often missing in many real-world datasets, of our synthesized images significantly reinforce the training process.

Figure 7b compares the three training strategies with another compilation of BDD100K images, which is composed of carefully chosen 200 rare images with unusual weather and lighting conditions such as rainy days, night driving, and driving in the sunset. This experiment is for estimating how the object detectors will perform in harsh driving environments. In the figure, note that with the same baseline training images, the object detection accuracy significantly deteriorates, from 62.55 to 37.77, in the harsh driving scenarios. The figure also shows that adding just a small number of synthesized images significantly improves the performance, whereas adding KITTI images hardly increases the object detection accuracy. The gap between **KITTI only** and **KITTI + CarFree (random)** is much more obvious than the results in Figure 7a. At the 1500 training images, **KITTI only**'s mAP is just 50.7, while **KITTI + CarFree (random)** reaches 73.65 and **KITTI + CarFree ($\geq$4 objects)** reaches 77.51. The results indicate that when the detection model is trained only with monotonous real-world images like KITTI, it does not perform well in harsh real-world driving scenarios. In such cases, adding CarFree's synthesized images with various weather conditions significantly improves the performance.

As evaluated above, CarFree can easily generate lots of training images and GTs with pixel-level precision without any human labor. Further, since we can freely control the weather and lighting conditions and artificially compose complex driving scenes in the virtual world, our synthesized images are of great diversity. Thus, our synthesized images

are more effective for the DNN training than the plain boring real-world driving images found in most public autonomous driving datasets.

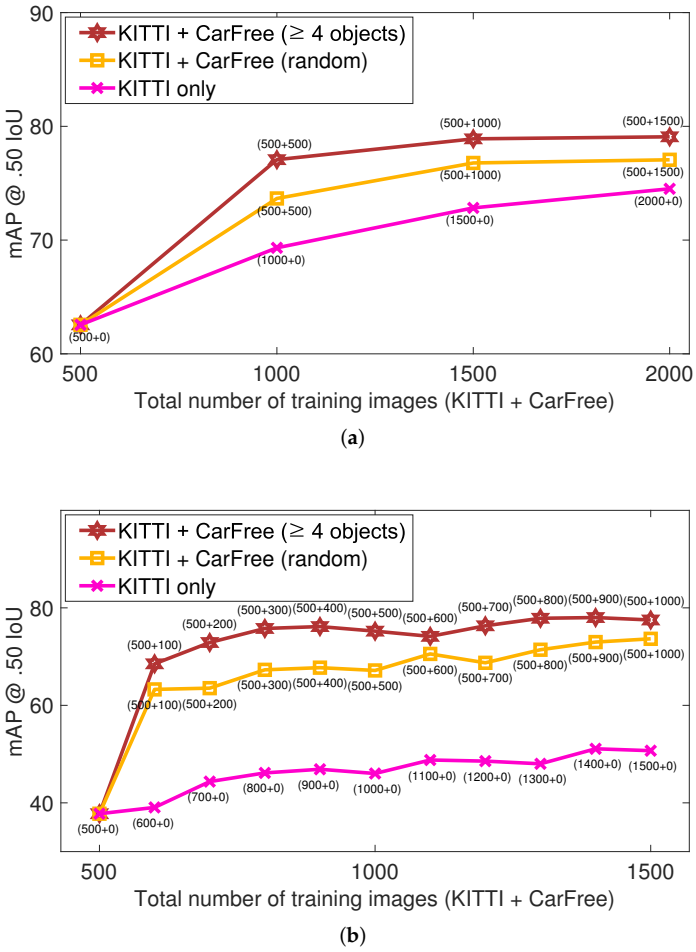


Figure 7. Evaluation of the training performance. At each result point, the mixture of KITTI and CarFree images is denoted by (number of KITTI images + number of CarFree images). (**a**) Evaluation results for usual driving conditions. (**b**) Evaluation results for harsh driving conditions.

7. Conclusions

This study's goal is to break the entry barrier for making training datasets of the object detection task in autonomous driving. Since gathering images with enough diversity and annotating their GTs require massive human labor, generating a good enough dataset is too costly. Our tool, CarFree, is based on the Carla autonomous driving simulator, under which we can freely simulate diverse driving scenarios and weather and lighting conditions. While the ego vehicle is driving around the virtual world, our developed Carla clients extract information from the Carla server later used by the post-processing GT generation program. Our GT generation method accepts the RGB images, the semantic segmentation images, and the 3D locations of vehicles and pedestrians in the simulation, and calculates the exact 2D location of each visible object in the forward-facing camera. All the process is fully automated, and the synthesized images are pixel-level precise for locating objects. Our evaluation results show that employing our synthesized images significantly improves the object detection performance of a state-of-the-art object detection model. Performance improvement is even more significant when the model is evaluated with test images exhibiting unusual weather conditions, which can significantly benefit the safety of autonomous driving systems operating in harsh environments.

In the future, we plan to update CarFree to make it more practical and efficient. First, we plan to extend CarFree such that it can generate other object labels such as lane markings and traffic signals. Second, to increase the throughput of the image generation process, we plan to develop a new feature that deploys multiple ego vehicles by running multiple separate Carla clients in parallel.

Author Contributions: Conceptualization, J.J. and J.-C.K.; methodology, J.J.; software, J.J.; validation, J.J., H.L. and J.-C.K.; formal analysis, J.J.; investigation, J.J.; resources, J.J.; data curation, J.J., H.L. and J.-C.K.; writing—original draft preparation, J.J.; writing—review and editing, J.-C.K.; visualization, J.J. and H.L.; supervision, J.-C.K.; project administration, J.-C.K.; funding acquisition, J.-C.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported partly by the Korea Evaluation Institute Of Industrial Technology (KEIT) grant funded by the Ministry of Trade, Industry and Energy (MOTIE) (20000316, Scene Understanding and Threat Assessment based on Deep Learning for Automatic Emergency Steering) and partly by the Transportation Logistics Development Program (21TLRP-B147674-04, Development of Operation Technology for V2X Truck Platooning) funded by the Ministry of Land, Infrastructure and Transport (MOLIT Korea).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ros, G.; Sellart, L.; Materzynska, J.; Vazquez, D.; Lopez, A.M. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3234–3243.
2. Gaidon, A.; Wang, Q.; Cabon, Y.; Vig, E. Virtual worlds as proxy for multi-object tracking analysis. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 4340–4349.
3. Tremblay, J.; Prakash, A.; Acuna, D.; Brophy, M.; Jampani, V.; Anil, C.; To, T.; Cameracci, E.; Boochoon, S.; Birchfield, S. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 969–977.
4. Cabon, Y.; Murray, N.; Humenberger, M. Virtual KITTI 2. *arXiv* **2020**, arXiv:2001.10773.
5. Johnson-Roberson, M.; Barto, C.; Mehta, R.; Sridhar, S.N.; Rosaen, K.; Vasudevan, R. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv* **2016**, arXiv:1610.01983.
6. Richter, S.R.; Hayder, Z.; Koltun, V. Playing for benchmarks. In Proceedings of the 16th IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 2232–2241.
7. NVIDIA. NVIDIA Drive Constellation: Virtual Reality Autonomous Vehicle Simulator. 2017. Available online: https://developer.nvidia.com/drive/drive-constellation (accessed on 26 July 2021).
8. Madrigal, A.C. Inside Waymo's Secret World for Training Self-Driving Cars. 2017. Available online: https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/ (accessed on 26 July 2021).
9. Cognata. Simulating Autonomy Autonomous Robotics Simulation Platform. 2018. Available online: https://www.cognata.com (accessed on 26 July 2021).
10. Rootman, S. Cognata Builds Cloud-Based Autonomous Vehicle Simulation Platform with NVIDIA and Microsoft. 2018. Available online: https://www.cognata.com/cognata-builds-cloud-based-autonomous-vehicle-simulation-platform-nvidia-microsoft (accessed on 26 July 2021).
11. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 1–16.
12. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *arXiv* **2017**, arXiv:1705.05065.
13. LG Electronics. A ROS/ROS2 Multi-Robot Simulator for Autonomous Vehicles. 2019. Available online: https://github.com/lgsvl/simulator (accessed on 26 July 2021).
14. NVIDIA. NVIDIA DRIVE Constellation. 2019. Available online: https://developer.nvidia.com/drive/drive-constellation (accessed on 26 July 2021).
15. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
16. Redmon, J. Darknet: Open Source Neural Networks in C. 2013–2016. Available online: http://pjreddie.com/darknet/(accessed on 26 July 2021).
17. Girshick, R. Fast R-CNN. In Proceedings of the 15th IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1440–1448.

18. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems (NeurIPS), Montreal, QC, Canada, 7–12 December 2015; pp. 91–99.
19. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single shot multibox detector. In Proceedings of the 14th European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 11–14 October 2016; pp. 21–37.
20. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
21. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. *arXiv* **2017**, arXiv:1704.05519.
22. Wu, Y.; Kirillov, A.; Massa, F.; Lo, W.Y.; Girshick, R. Detectron2. 2019. Available online: https://github.com/facebookresearch/detectron2 (accessed on 11 August 2021).
23. Lin, S.C.; Zhang, Y.; Hsu, C.H.; Skach, M.; Haque, M.E.; Tang, L.; Mars, J. The architectural implications of autonomous driving: Constraints and acceleration. In Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Williamsburg, VA, USA, 24–28 March 2018; pp. 751–766.
24. Kato, S.; Tokunaga, S.; Maruyama, Y.; Maeda, S.; Hirabayashi, M.; Kitsukawa, Y.; Monrroy, A.; Ando, T.; Fujii, Y.; Azumi, T. Autoware on board: Enabling autonomous vehicles with embedded systems. In Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018; pp. 287–296.
25. Alcon, M.; Tabani, H.; Kosmidis, L.; Mezzetti, E.; Abella, J.; Cazorla, F.J. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In Proceedings of the 26th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Sydney, NSW, Australia, 21–24 April 2020; pp. 267–280.
26. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 22nd IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Miami, FL, USA, 20–25 June 2009; pp. 248–255.
27. Everingham, M.; Van Gool, L.; Williams, C.K.; Winn, J.; Zisserman, A. The PASCAL visual object classes challenge: A retrospective. *Int. J. Comput. Vis. (IJCV)* **2015**, *111*, 98–136. [CrossRef]
28. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the 13rd European Conference on Computer Vision (ECCV), Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
29. Janai, J.; Güney, F.; Behl, A.; Geiger, A. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *arXiv* **2017**, arXiv:1704.05519.
30. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for autonomous driving? the KITTI vision benchmark suite. In Proceedings of the 25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, USA, 16–21 June 2012; pp. 3354–3361.
31. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3213–3223.
32. Neuhold, G.; Ollmann, T.; Rota Bulo, S.; Kontschieder, P. The mapillary vistas dataset for semantic understanding of street scenes. In Proceedings of the 16th IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 4990–4999.
33. Yu, F.; Chen, H.; Wang, X.; Xian, W.; Chen, Y.; Liu, F.; Madhavan, V.; Darrell, T. BDD100K: A diverse driving dataset for heterogeneous multitask learning. In Proceedings of the 33rd IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020; pp. 2636–2645.
34. Chang, M.F.; Lambert, J.W.; Sangkloy, P.; Singh, J.; Bak, S.; Hartnett, A.; Wang, D.; Carr, P.; Lucey, S.; Ramanan, D.; Hays, J. Argoverse: 3D tracking and forecasting with rich maps. *arXiv* **2019**, arXiv:1911.02620.
35. Huang, X.; Cheng, X.; Geng, Q.; Cao, B.; Zhou, D.; Wang, P.; Lin, Y.; Yang, R. The apolloscape dataset for autonomous driving. In Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 954–960.
36. Caesar, H.; Bankiti, V.; Lang, A.H.; Vora, S.; Liong, V.E.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; Beijbom, O. nuscenes: A multimodal dataset for autonomous driving. *arXiv* **2019**, arXiv:1903.11027.
37. Sun, P.; Kretzschmar, H.; Dotiwalla, X.; Chouard, A.; Patnaik, V.; Tsui, P.; Guo, J.; Zhou, Y.; Chai, Y.; Caine, B.; et al. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv* **2019**, arXiv:1912.04838.
38. Kesten, R.; Usman, M.; Houston, J.; Pandya, T.; Nadhamuni, K.; Ferreira, A.; Yuan, M.; Low, B.; Jain, A.; Ondruska, P.; et al. Lyft Level 5 Perception Dataset 2020. 2019. Available online: https://level5.lyft.com/dataset/ (accessed on 11 August 2021).
39. Unity. 2020. Available online: https://unity.com/ (accessed on 11 August 2021).
40. Epic Games. Unreal Engine. 2020. Available online: https://www.unrealengine.com (accessed on 11 August 2021).
41. Richter, S.R.; Vineet, V.; Roth, S.; Koltun, V. Playing for data: Ground truth from computer games. In Proceedings of the 14th European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 11–14 October 2016; pp. 102–118.
42. Su, H.; Deng, J.; Fei-Fei, L. Crowdsourcing annotations for visual object detection. In Proceedings of the 26th AAAI Conference on Artificial Intelligence (AI), Toronto, ON, Canada, 22–126 July 2012.
43. OpenCV. cvat. 2018. Available online: https://github.com/opencv/cvat (accessed on 11 August 2021).
44. Microsoft. VoTT. 2018. Available online: https://github.com/microsoft/VoTT (accessed on 11 August 2021).
45. Brooks, J. COCO Annotator. 2019. Available online: https://github.com/jsbroks/coco-annotator/ (accessed on 11 August 2021).

46. Dutta, A.; Zisserman, A. The VIA annotation software for images, audio and video. In Proceedings of the 27th ACM International Conference on Multimedia, Nice, France, 21–25 October 2019; pp. 2276–2279.
47. AlexeyAB. GUI for Marking Bounded Boxes of Objects in Images for Training Neural Network Yolo v3 and v2. 2016. Available online: https://github.com/AlexeyAB/Yolo_mark (accessed on 11 August 2021).
48. Wada, K. labelme: Image Polygonal Annotation with Python. 2016. Available online: https://github.com/wkentaro/labelme (accessed on 11 August 2021).
49. Castrejon, L.; Kundu, K.; Urtasun, R.; Fidler, S. Annotating object instances with a polygon-rnn. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5230–5238.
50. Papadopoulos, D.P.; Uijlings, J.R.; Keller, F.; Ferrari, V. Extreme clicking for efficient object annotation. In Proceedings of the 16th IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 4930–4939.
51. Acuna, D.; Ling, H.; Kar, A.; Fidler, S. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 859–868.
52. Rhinehart, N.; McAllister, R.; Kitani, K.; Levine, S. Precog: Prediction conditioned on goals in visual multi-agent settings. In Proceedings of the 17th IEEE International Conference on Computer Vision (ICCV), Seoul, Korea, 27–28 October 2019; pp. 2821–2830.
53. Kaneko, M.; Iwami, K.; Ogawa, T.; Yamasaki, T.; Aizawa, K. Mask-SLAM: Robust feature-based monocular SLAM by masking using semantic segmentation. In Proceedings of the 31st IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 258–266.
54. Ravi Kiran, B.; Roldao, L.; Irastorza, B.; Verastegui, R.; Suss, S.; Yogamani, S.; Talpaert, V.; Lepoutre, A.; Trehard, G. Real-time dynamic object detection for autonomous driving using prior 3D-maps. *arXiv* **2018**, arXiv:1809.11036.
55. Pan, X.; You, Y.; Wang, Z.; Lu, C. Virtual to real reinforcement learning for autonomous driving. *arXiv* **2017**, arXiv:1704.03952.
56. Sallab, A.E.; Abdou, M.; Perot, E.; Yogamani, S. Deep reinforcement learning framework for autonomous driving. *arXiv* **2017**, arXiv:1704.02532.
57. Liang, X.; Wang, T.; Yang, L.; Xing, E. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In Proceedings of the 15th European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 584–599.
58. Hanhirova, J.; Debner, A.; Hyyppä, M.; Hirvisalo, V. A machine learning environment for evaluating autonomous driving software. *arXiv* **2020**, arXiv:2003.03576.
59. Karpaphy, A. [CVPR'21 WAD] Keynote—Andrej Karpathy, Tesla. 2021. Available online: https://youtu.be/g6bOwQdCJrc (accessed on 27 August 2021).