



Technische Universität München  
TUM School of Engineering and Design  
Photogrammetrie und Fernerkundung  
Prof. Dr.-Ing. U. Stilla

# Vehicle detection in aerial images using neural networks with synthetic training data


**Shuangyi Liu**

Master's Thesis

**Submission:** 01.05.2022 - 01.11.2022

**Study Course:** Earth Oriented Space Science and Technology (Master)

**Supervisors:** Olaf Wysocki  
Corentin Henry (DLR)  
Dr. Nina Merkle (DLR)  
Dr. Seyed Majid Azimi (DLR)  
Manuel Mühlhaus (DLR)

**In Cooperation with:** Deutsches Zentrum für Luft- und Raumfahrt  
 Institut für Methodik der Fernerkundung  
Photogrammetrie und Bildanalyse

2022

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, Submission date

Shuangyi Liu

## Acknowledgments

First and foremost, I would like to thank you to my Master's thesis supervisors, Olaf Wysocki, Corentin Henry, Dr. Nina Merkle, and Dr. Seyed Majid Azimi. I feel extremely lucky to have so many supervisors from both institutes of Technische Universität München (TUM) and Deutsches Zentrum für Luft- und Raumfahrt (DLR) to guide and assist me throughout my whole thesis journey. Due to your support and dedicated involvement in every step of my thesis, I am able to achieve this research. I learned so much from you in terms of both academic knowledge and the scientific spirits of being dedicated, patient and detailed. Thanks to Manuel Mühlhaus for sharing data and his knowledge in object detection. Thank you to Yuxing Xie for being so supportive and helpful.

It is very much appreciated for having this wonderful opportunity to write my master thesis at Photogrammetry and Image Analysis department at DLR. Thanks to everyone I worked with and met during the short six months of working there: your friendship and support have added a lot to my life.

Thanks of course to my family and dear friends, Ruchen Tian and Ge Shi, for making my life full of laughter even during hard times.

# Abstract

Deep learning approaches have made great strides in pattern recognition due to their superior performance. Such approaches require a large amount of ground-truth data. However, it is a challenge to collect enough real training data and label them manually due to cost and time consumption. To overcome this problem, an alternative approach is to use synthetic data with automatically generated ground truth. By means of synthetic data generation, large amount of images can be extracted directly from a virtual scene. These simulated images can be customized according to the specific needs of the use-case. Therefore, this thesis focuses on the use of synthetic data in vehicle detection.

A pipeline for generating synthetic data is based on the real-time 3D creation tool Unreal Engine and the drone simulator AirSim. Unreal Engine provides a simulation environment that allows one to simulate complex situations in a virtual world, such as data acquisition with drones. AirSim on the other hand, is a simulator for drones and cars, which works as a plugin for the Unreal Engine. By accessing the AirSim Application Programming Interfaces (APIs), we can retrieve images from a virtual scene at desired camera locations. An existing virtual scene, City Park Environment Collection, and multiple vehicle assets are downloaded and used. The main focus for this thesis is vehicle placement in the virtual scene with the goal of creating realistic scenarios. The scenarios include generating various traffic situations, such as traffic jam, normal traffic flow, and sparsely placed vehicles.

The object detector used in this thesis is the Faster R-CNN detector (ReDet). The detection performance of experiments trained with a variety of training and testing datasets are compared and evaluated quantitatively and qualitatively. The evaluation metrics used are true positive rate, false negative rate, average precision, and precision recall curves. Furthermore, we provide insights about the domain similarity of each of these datasets and the effects of having a limited amount of real-world data for training.

The results indicate that a domain gap exists between the synthetic and real-world data, yet with an increasing ratio of real-world data for training, the detection performance can be comparable to which trained with only real-world images. The synthetic

---

images generation pipeline can be a guidance or implemented directly for future synthetic image generation. The experimental results of this thesis can be used as a reference when researching on the detection performance using mixed synthetic and real training data.

# Contents

<b>Acknowledgments</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Challenges . . . . .	2
1.3 Contributions . . . . .	3
1.4 Thesis outline . . . . .	3
<b>2 State of the Art</b>	<b>4</b>
2.1 Traditional detection methods . . . . .	4
2.1.1 Haar wavelets . . . . .	4
2.1.2 HOG Detector . . . . .	4
2.1.3 Deformable part-based models (DPM) . . . . .	4
2.2 Machine learning based detection methods . . . . .	5
2.2.1 Bag-of-words (BoW) . . . . .	6
2.2.2 Support vector machine (SVM) . . . . .	6
2.2.3 AdaBoost . . . . .	6
2.3 Deep learning based detection methods . . . . .	6
2.3.1 Convolutional Neural Networks (CNNs) . . . . .	7
2.3.2 Two-stage CNN . . . . .	8
2.3.3 Single-stage CNN . . . . .	10
2.4 Real-world datasets for vehicle detection . . . . .	10
2.4.1 VEDAI . . . . .	11
2.4.2 DOTA . . . . .	12
2.4.3 EAGLE . . . . .	13
2.5 Synthetic training data for vehicle detection . . . . .	14
2.5.1 Methods of generating synthetic data . . . . .	14
2.5.2 Synthetic datasets generated from game engines . . . . .	15
2.6 Virtual scene and drone/car simulator . . . . .	16
2.6.1 Simulation platforms . . . . .	16
2.6.2 Available 3D city models for simulation . . . . .	18

2.6.3	Drone/car simulator . . . . .	19
<b>3</b>	<b>Methodology</b>	<b>22</b>
3.1	Vehicle assets . . . . .	23
3.1.1	Creating vehicles with various colors . . . . .	23
3.2	Vehicle placement . . . . .	24
3.2.1	Obtain initial vehicle position . . . . .	24
3.2.2	Extract orientation information . . . . .	25
3.2.3	Traffic scene generation . . . . .	29
3.3	Segmented images generation . . . . .	33
3.3.1	AirSim set up . . . . .	33
3.3.2	Instance segmentation . . . . .	33
3.4	Bounding boxes generation . . . . .	34
3.5	Training using ReDet detector . . . . .	36
3.5.1	ReDet detector . . . . .	36
<b>4</b>	<b>Evaluation and Analysis</b>	<b>38</b>
4.1	Experiment set up . . . . .	38
4.1.1	Synthetic datasets . . . . .	38
4.1.2	EAGLE dataset used for training and testing . . . . .	38
4.1.3	Training with ReDet . . . . .	39
4.1.4	Experiments design . . . . .	39
4.2	Vehicle detection metrics . . . . .	42
4.2.1	True Positive Rate and False Negative Rate . . . . .	42
4.2.2	Precision and Recall . . . . .	42
4.2.3	Precision-Recall curve . . . . .	43
4.2.4	Intersection over Union (IoU) . . . . .	43
4.2.5	Average Precision (AP) . . . . .	43
4.3	Realistic scenario setting . . . . .	44
4.4	Domain Gap . . . . .	45
4.5	Synthetic and real-world data mixing . . . . .	49
<b>5</b>	<b>Discussion</b>	<b>53</b>
<b>6</b>	<b>Conclusions and Outlook</b>	<b>56</b>
	<b>List of Figures</b>	<b>58</b>
	<b>List of Tables</b>	<b>61</b>

*Contents*

---

**Bibliography**

**62**



# 1 Introduction

*Artificial intelligence* (AI) is defined in the field of AI research as the computer that perceives its environment and takes actions that maximize its chance of success at some goals [1]. *Machine learning* is a subfield of AI technology, and involves research into how the computer simulates or realizes human behavior to gain new knowledge or skills, and then continually improves its own performance [2]. *Deep learning* is a necessary tool to achieve machine learning for artificial intelligence. Based on the machine's increased calculation ability and the emergence of big data era, deep learning enables the computer to learn by increasing the number of layers of networks [2]. In recent years, deep learning is one of the most effective methods in the field of computer vision such as automatic driving, face recognition, object detection, object tracking, etc [2].

Object detection is an important computer vision task that deals with detecting instances of visual objects of a certain class (vehicle in our case) in digital images [3]. Vehicle detection from aerial images is becoming an increasingly important research topic in surveillance, traffic monitoring and military applications [4]. In this thesis, we focus on using simulated data to train neural networks to detect vehicles.

## 1.1 Motivation

Vehicle detection based on aerial imagery is crucial for a variety of applications such as large-scale traffic monitoring, parking lot utilization, urban planning, disaster management as well as search and rescue missions [5]. The wide field of view of aerial images provides valuable information over large open areas in a short time [6].

Vehicle detection benefits greatly from deep neural networks. The major success of these models can be attributed to the availability of required computational power to perform massive calculations, and to the availability of large datasets to learn the transformation functions [7]. However, the collection of raw data, annotation and verification of these large datasets is an expensive and time-consuming task. On the contrary, simulated data can be automatically generated in large quantity and customized in terms of resolution, image sizes, camera parameters, etc. based on the

specific use-case. When building a high quality dataset, generating simulated data can be a promising method to introduce cost saving measures.

## 1.2 Challenges

Synthetic data can be generated by using modern video games [8] or 3D game engines [9–12]. These approaches allow extracting a large amount of accurate ground truth data, such as bounding boxes, segmentation maps, depth maps or camera poses, as this information is already available in the virtual world from which data is rendered.

A few challenges for creating simulated data can be as follows:

- Realistic scenario settings

In order to generate synthetic dataset that is comparable to real-world dataset, it must be qualified with measuring factors that could impact the effectiveness of the dataset. To create realistic settings of vehicle placement, observations should be made in the real world and the scenarios in the virtual world should imitate which in the real world.

- Occlusions problem

Occlusion of vehicles by other objects in the camera's viewpoint results in partial visibility in rendered images. This leads to the scenario of partially visible cars with the detection algorithm.

- Parametric simulation environment

Synthetic datasets contain images with different orientation of cameras, light conditions, camera positions and parameters, such as exposure and field of view. These variables have an important impact on how objects are depicted in the image and consequently on the object detection accuracy. However, it requires a large amount of experiments to discover the best-suited variables.

- Synthetic image realism

Another challenge is to make the synthetic images look as realistic as possible in terms of the details of the texture, shape, color of the objects in the simulation environment. This objective involves the virtual world design and the image noises.

- Vehicle placement

Based on the City Park Environment Collection, there is limited semantic information of road system. The position and orientation of the road are not provided directly from the virtual world. An algorithm must be designed to extract the semantic information needed for placing vehicles on the road.

## 1.3 Contributions

This study focuses on the generation of synthetic training data, training a predefined model, and evaluating the performance based on different training datasets. The contributions can be listed as follows:

- Automatically placing vehicles in a simulated environment to imitate realistic scenarios.
- Presenting a pipeline of synthetic image generation with adjustable image size, camera parameters (i.e., exposure, field of view), camera position and orientation.
- Providing a detailed insight into the performance of a deep learning-based vehicle detection method on synthetic images.
- Studying how different ratios of real to synthetic images in the training dataset can achieve comparable results with respect to only real-world images.

## 1.4 Thesis outline

Chapter 2 reviews the state of the art in the field of related work. Chapter 3 explains the method used in this thesis, such as creating the virtual scene, generating synthetic data, and training with neural networks. The results of vehicle detection with different training datasets are evaluated and analysed in Chapter 4 and discussed in Chapter 5. The conclusion and future work of this thesis are presented in Chapter 6.

## 2 State of the Art

### 2.1 Traditional detection methods

Most of the early object detection algorithms were built based on the features extracted from the object. Handcrafted features are used to determine the performance of methods.

#### 2.1.1 Haar wavelets

Haar wavelets identify local, oriented intensity-difference features at different scales [13]. With Haar wavelets, the relationship between average intensities of neighboring regions can be captured for an image representation [13]. A Haar-like feature is the result of integrating a wavelet with a patch [14], which has become an increasingly indispensable tool for extracting information in the field of object detection [15]. Haar-like features have been widely used in face detection [14], pedestrian detection [16], vehicle detection [17], etc.

#### 2.1.2 HOG Detector

Histogram of Oriented Gradient (HOG) was introduced in 2005 by N. Dalal and B. Triggs [18]. To balance the feature invariance (including translation, scale, illumination, etc) and the nonlinearity (on discriminating different objects categories), the HOG descriptor is designed to be computed on a dense grid of uniformly spaced cells and use overlapping local contrast normalization for improving accuracy [3]. To detect objects of different sizes, the HOG detector rescales the input image for multiple times while keeping the size of a detection window unchanged [3]. The HOG detector has been an important foundation of many object detectors [19, 20] and a large variety of computer vision applications for many years [3].

#### 2.1.3 Deformable part-based models (DPM)

DPM was originally proposed by P. Felzenszwalb [21] in 2008 as an extension of the HOG detector. The idea of DPM is to learn the proper way of decomposing an object. For example, the detection of a vehicle can be considered as the detection of its window,

body, and wheels. A typical DPM detector consists of a root-filter and a number of part-filters [3]. The configurations of part filters (e.g., size and location) can be learned automatically as latent variables under a weakly supervised learning method [3]. The detection is later sped up with the implementation of a cascade architecture developed by R. Girshick [22, 23].

## 2.2 Machine learning based detection methods

The progress of using handcrafted features has slowed down after 2010 [3], because traditional methods cannot reach an optimal balance between the discriminability and the robustness without considering the details of real-world data [24]. Machine learning has gained popularity in the photogrammetry and computer vision community. With the powerful feature representations and classifiers, machine learning techniques has made significant improvement in object detection. Figure 2.1 gives the flowchart of machine learning-based object detection, in which object detection can be performed by learning a classifier that captures the variation in object appearances and views form a set of training data in a supervised or semi-supervised or weakly supervised framework [25]. The input of the classifier is a set of regions (sliding windows or object proposals) with their corresponding feature representations and the output is their corresponding predicted labels, i.e., object or not [25]. As seen in Figure 2.1, feature extraction, feature fusion and dimension reduction, and classifier training play the most important roles in the performance of object detection [25].

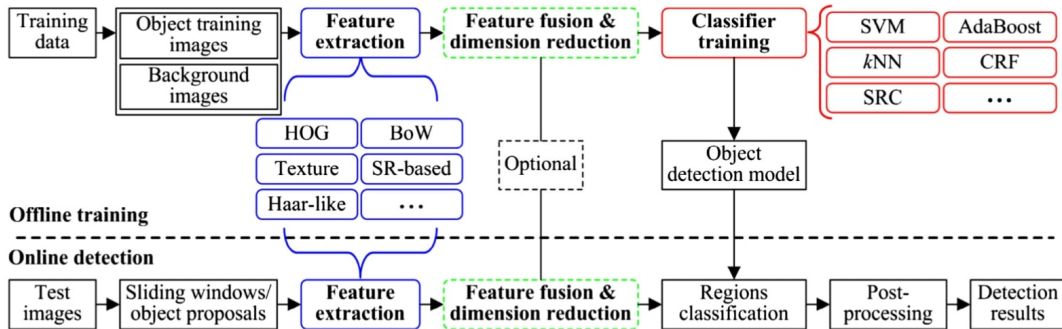


Figure 2.1: The flowchart of machine learning-based object detection [25].

### 2.2.1 Bag-of-words (BoW)

Bag-of-words (BoW) [26] quantize local descriptors into "visual words", in which way, each image can be viewed as a long and sparse vector of words [27]. By applying scalable indexing and fast search on this vector space, text-retrieval system can be imitated [27]. Zhou et al. [28] apply a Bag-of-words (BoW) model and a local steering kernel with the sliding window strategy to detect vehicles in arbitrary orientations. They utilize a collection of representative features associated with various entities, including windshield, roof, and hood [28].

The main advantages of the method are the simplicity, efficiency and invariance even under viewpoint changes and background clutter [25]. One major concern is that BoW model could not support spatial properties of local descriptors of images [27]. Another disadvantage is that key points are extracted from the grey level images and local descriptors do not contain any color information [29]. Moreover, they only grasp the local information, losing the overall distribution of visual information [27].

### 2.2.2 Support vector machine (SVM)

Support vector machine (SVM) [30] is one of the most popular and effective machine learning algorithms for solving classification problems [25]. In its simplest form, SVMs are linear binary classifiers that assign a given test sample a class from one of the two possible labels [31]. Furthermore, SVM can also be used as a non-linear classifier (called kernel SVM) to further improve the separation between classes by projecting the samples onto a higher dimensional feature space [25].

### 2.2.3 AdaBoost

The Adaboost algorithm [32, 33] is a widely used machine learning algorithm that combines many weak classifiers to form a strong classifier by adjusting the weights of training samples [25]. The AdaBoost classifier has shown good performance in many object detection applications such as vehicle detection [17, 19].

## 2.3 Deep learning based detection methods

Deep learning techniques refer to a cluster of machine learning methods that construct a multilayered representation of the input data [34]. There are several deep learning methods. Object detection started to evolve at an unprecedented pace, since deep convolutional neural network (CNN) has been brought in.

### 2.3.1 Convolutional Neural Networks (CNNs)

CNNs are a region proposal deep learning object detection technique. CNNs are the base for other common object detection models like R-CNN, Fast R-CNN, Faster R-CNN. The structure of a general CNN architecture is shown in Figure 2.2.

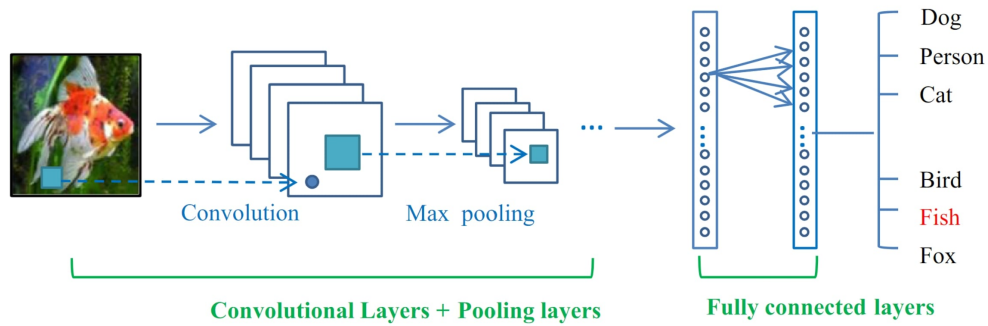


Figure 2.2: The pipeline of general CNN architecture [35]

The three main aspects of CNN are: convolutional layer, pooling layer, and fully connected (FC) layer. Filters are made up of pre-defined kernels of a certain matrix size. They run across a matrix of image pixels, which is referred to as the input feature map. The multiplication of the kernel and input feature map form an output matrix. For each convolution, each neuron in a hidden layer will compute the weighted sum of its inputs and apply bias and activate with a local non-linearity. Each neuron in the hidden layer is only seeing a patch from the original input image.

The CNN forms intermediate feature maps as well as feature maps per whole image throughout the process. The pooling layer computation reduces the size of the feature map via average pooling and max pooling. They are a strategy used to learn higher order features like shapes or specific objects within the layers of the model. The FC layers convert 2D feature maps into 1D feature vectors. This is the step where the model uses the vector to identify the object.

The combination of convolutional layers and other layering techniques defines a model. Deep learning is to used the layer of operations to learn the hierarchy of features. Common CNNs for object detection are Alexnet, ResNet and VGG.

Several CNN methods in deep learning have been proposed for vehicle detection. In the following subsections, we discuss two directions of CNN methods: single-stage and two-stage.

### 2.3.2 Two-stage CNN

For two-stage vehicle detector, the first stage is to extract features from raw image data and second stage is to make final predictions.

#### Region-based convolutional neural network (R-CNN)

The first method is called R-CNN [22], which achieved good performance in vehicle detection. R-CNN combines region proposal network with CNN, achieving better performance compared to HOG features [20] with a SVM classifier. The architecture of R-CNN models is shown in Figure 2.3.

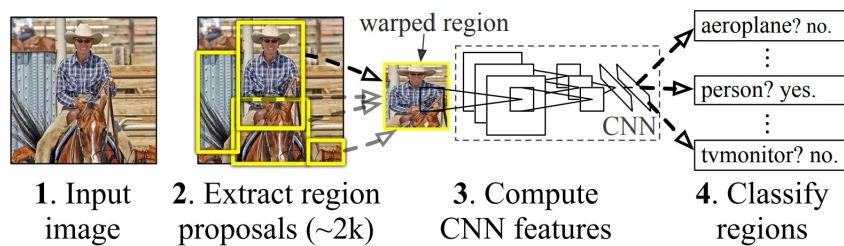


Figure 2.3: R-CNN architecture [22].

A R-CNN takes raw image data as an input through a selective search and extracts region proposals. Each proposal is rescaled to a fixed size image. Then, the region proposals are fed into a CNN to extract features, and a SVM is used to make predictions [36]. The primitive R-CNN achieved 53% mean average precision in the Pascal VOC 2010 competition. Although R-CNN has made great progress, its drawback is that extracting a large amount of features leads to an extremely slow detection speed [3]. Spatial Pyramid Pooling (SPP) [37] applies a convolution layer to the entire image and extracts features using SPP-net, which addresses the expensive computational cost of R-CNN.

#### Fast R-CNN

Fast R-CNN, presented in [23], is built like an R-CNN model, but the main difference is that instead of extracting features independently for each region proposal, the model aggregates them into one CNN forward pass over the entire image and the region proposals share this feature matrix. The architecture of Fast R-CNN model is depicted in Figure 2.4.

The benefit of this model is that the training can update all network layers including the convolutional layers before the Region of Interest (RoI) pooling layers. The RoI



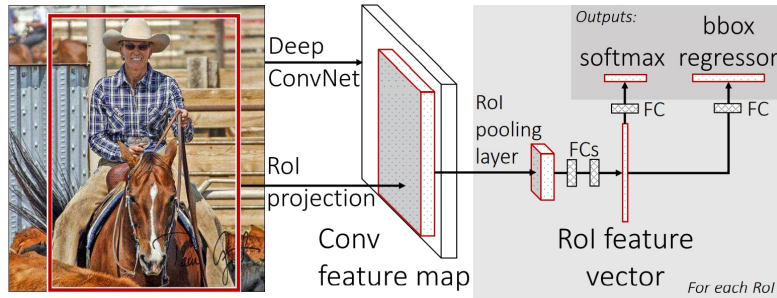


Figure 2.4: Fast R-CNN architecture [23].

pooling layer replaces the last max pooling layer of the pre-trained CNN. The pooling layer outputs a fixed length feature vectors that is fed through the last fully connected layer to a bounding box regression model to predict classes.

### Faster R-CNN

Shortly after Fast R-CNN was proposed, faster R-CNN was proposed [38]. This model composed of the region proposal network (RPN) and Fast R-CNN meshed into one model with shared convolutional feature layers. The architecture of Faster R-CNN model is depicted in Figure 2.5.

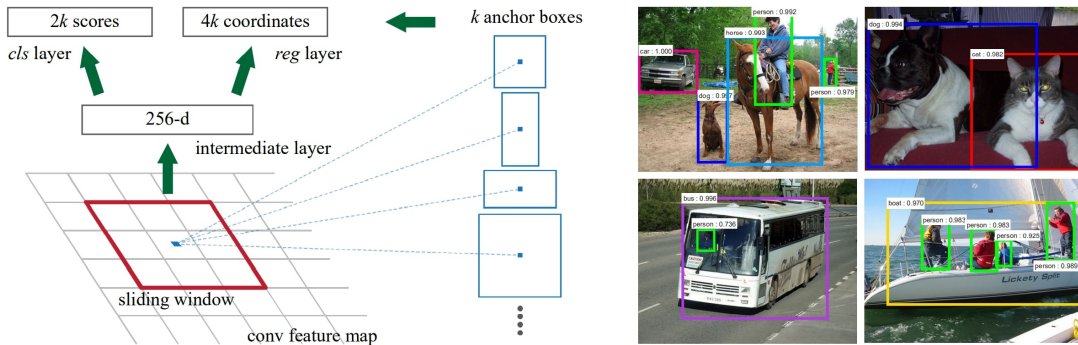


Figure 2.5: Faster R-CNN architecture and RPN [38]

The model pre-trained a CNN network on image classification task and fine tunes the region proposal network by sliding a  $n \times n$  matrix window over the convolution feature map of the entire image. The number of region proposals for each location is denoted as  $k$  anchor boxes. These proposals are then used to train the Fast R-CNN

object detection model. Then, the network uses the Fast R-CNN network to initialize RPN training process and fine tunes the unique layers of the Fast R-CNN. Then it is fed through the last fully connected layer to a SoftMax estimator and a bounding box regression model to predict classes.

### 2.3.3 Single-stage CNN

In order to improve the vehicle detection speed, single-stage methods have been proposed. Single-shot detector (SSD) [39] is one of the state-of-the-art single-stage detectors, which make predictions by utilizing different resolutions of feature maps. You Only Look Once (YOLO) [40] is another type of single-stage detector (Figure 2.6). YOLO regards raw image data as a  $7 \times 7$  grid and passes it only once in a fully convolutional neural network (F-CNN), which makes it quite fast and real-time. This network splits the image into regions and predicts bounding boxes and probabilities for each region simultaneously.

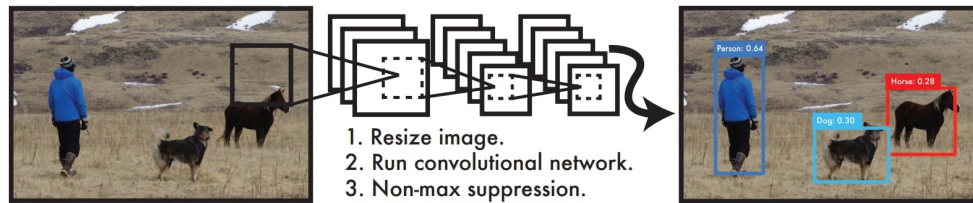


Figure 2.6: The YOLO detection system [40]

Later, Redmon et al. made a series of improvements on the basis of YOLO and proposed its v2 and v3 editions [41, 42], which further improves the detection accuracy while keeping a very high detection speed. Benjedira et al. [36] presented a performance evaluation of Faster R-CNN and YOLOv3 algorithms on car detection from aerial images, in terms of accuracy and processing time. Their study revealed that YOLOv3 outperforms Faster R-CNN in both car extraction with 99.07 % accuracy and processing time for one image detection [36].

## 2.4 Real-world datasets for vehicle detection

Datasets have played an important role in data-driven research in recent years. Large datasets like MSCOCO [43] are instrumental in promoting object detection and image captioning research [44]. ImageNet [45] () is popular for classification and scene

recognition task. For the purpose of remote object tracking and unmanned driving, datasets like VEDAI [46], DOTA [44], EAGLE [5] can be used for aerial object detection.

### 2.4.1 VEDAI

VEDAI (Vehicle Detection in Aerial Imagery) is a dataset designed to address the task of small vehicle detection in aerial images within a realistic industrial framework [46]. This dataset was made to help the development of new algorithms for aerial multi-class vehicle detection in unconstrained environment [46].

A total of 1210 images with a resolution of 1024x1024 pixels have been manually selected and included into 4 different sub-sets: large-size color images, small-size (downscaled) color images, large-size infrared images, and small-size (downscaled) infrared images [46]. The dataset contains many different types of backgrounds (e.e., fields, grass, mountains, urban area, etc.) and isolated vehicles to avoid false alarms [46].



Figure 2.7: Images illustrating the different categories of the dataset. From left to right: car, truck, camping car, tractor, plane, boat, other, pickup and van [46].

There are nine different classes of vehicles contained in the dataset, namely 'plane', 'boat', 'camping car', 'car', 'pick-up', 'tractor', 'truck', 'van', and the 'other' category [46]. Figure 2.7 illustrates these classes.

## 2.4.2 DOTA

DOTA is a large-scale Dataset for Object deTection in Aerial images [44]. The dataset consists of 2806 aerial images from different sensors and platforms [44]. Each image is of the size about 4000 × 4000 pixels and contains objects exhibiting a wide variety of scales, orientations, and shapes [44]. These DOTA images are annotated using 15 common object categories, containing 188,282 instances [44]. Some samples of annotated patches are shown in Figure 2.8.

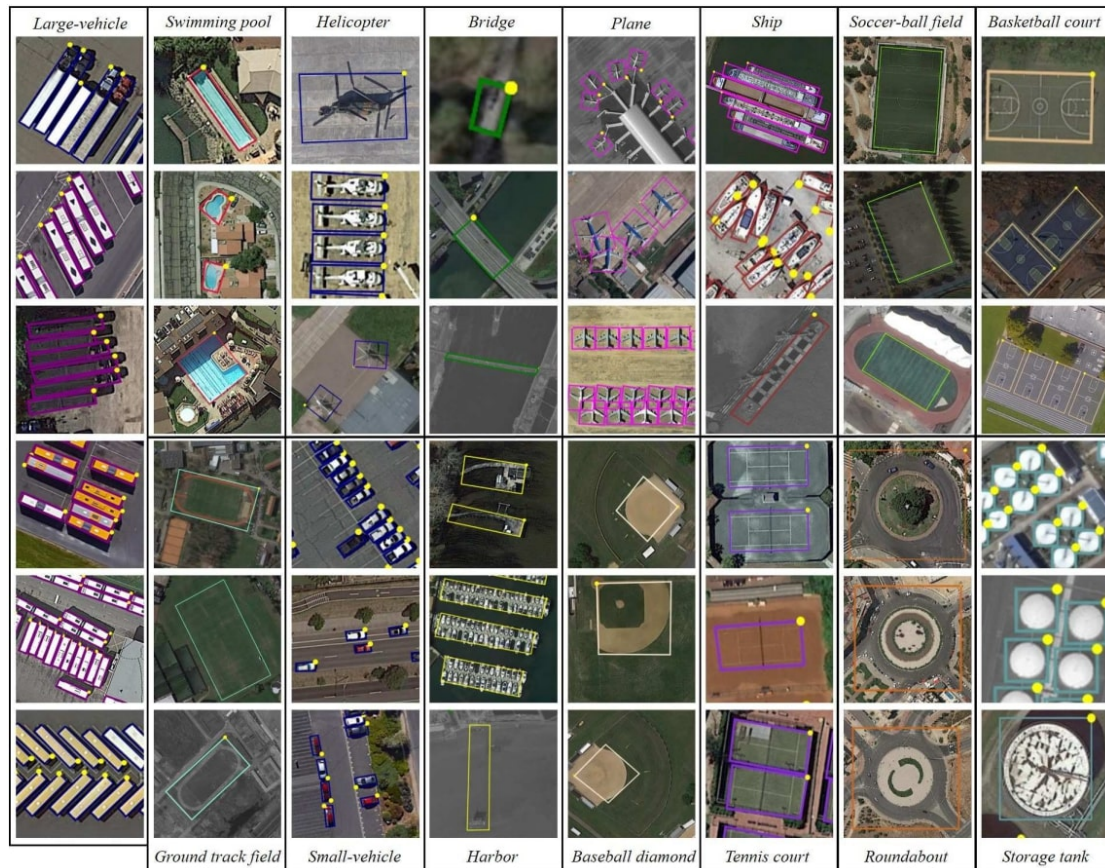


Figure 2.8: Samples of annotated images in DOTA. There are three samples for each category, except six for *large-vehicle*. [44]

### 2.4.3 EAGLE

EAGLE (oriEnted vehicle detection using Aerial imaGery in real-world scEnarios) is a large-scale dataset for multi-class vehicle detection with object orientation information in aerial imagery [5].

EAGLE dataset contains approximately 200 thousands annotated vehicles, ranging from 1 to 3.5 thousand annotations per image in all possible orientations (Figure 2.9). There are 345 original sized images of 5616 × 3722px size, which are tiled up to 8000 images with the size of 1024 × 1024px.

The GSDs of EAGLE dataset range from 5 cm to 45 cm per pixel. The images were taken from early in the morning until the evening in various weather conditions (e.g., sunny, snowy, rainy, and foggy) with different illumination levels [5]. Altogether, the variability in image parameters and scenes allows the dataset to cover a wide range of real-world situations involving vehicles.



Figure 2.9: Examples of annotated images [5]

## 2.5 Synthetic training data for vehicle detection

Due to the possibility of having diversified samples with automatically generated annotations, the use of realistic synthetic images has increased considerably in recent years within the photogrammetry and computer vision community.

### 2.5.1 Methods of generating synthetic data

#### Domain randomization

Domain randomization is an effective technique for data augmentation [47]. It is commonly done by generating new data that can simulate complex environments based on the limited training samples [47]. Manual transformations are usually implemented, such as: altering the location and texture of objects, changing the number and shape of objects, modifying the illumination and camera view, and adding different types of random noise to the data [47]. Many studies use real scenes and augments them with the synthetic objects [12, 48, 49].

#### GAN-based synthetic image generation

One approach inspired by game theory for training a model that synthesizes images is known as Generative Adversarial Networks (GANs) [50]. The model consists of two networks that are trained in an adversarial process where one network generates fake images and the other network discriminates between the real and fake images repeatedly [51]. As the generator improves with training, the discriminator performance gets worse because the discriminator cannot easily tell the difference between real and fake [52]. If the generator succeeds perfectly, then the discriminator has a 50% accuracy, reaching the convergence [52]. GANs have been used for generating remote sensing images [52, 53] for training deep neural networks for surveillance and scene understanding purposes.

#### Synthetic data generation through simulation

Synthetic data can be generated as frames from sophisticated environmental modeling [54–56] or as pictures taken from camera viewpoints in a simulation environment [57, 58]. Johnson-Roberson et al. [59] directly capture frames from the video games GTA 5 along with auxillary information that allows them to compute the label. Ros et al. [60] use computer graphics to generate urban scene images with perfect label. Richter et al. [8] produced dense pixel-level semantic annotations for 25 thousand images synthesized by a photo-realistic open-world computer game.

### 2.5.2 Synthetic datasets generated from game engines

Most works addressing synthetic data generation focus on driving, simulating constrained traffic situations, such as Synscapes (7D) [55], and Virtual KITTI [61]. Few works consider the generation of synthetic data captured from Unmanned Aerial Vehicles (UAVs). The following synthetic datasets address aerial images.

#### SYNTHIA

SYNTHIA is a synthetic collection of diverse urban images, with automatically generated class annotations [60]. This dataset consists of photo-realistic frames rendered from a virtual city using the Unity Engine and comes with precise pixel-level semantic annotations for 13 classes, i.e., sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists and miscellaneous (Figure 2.10) [60].

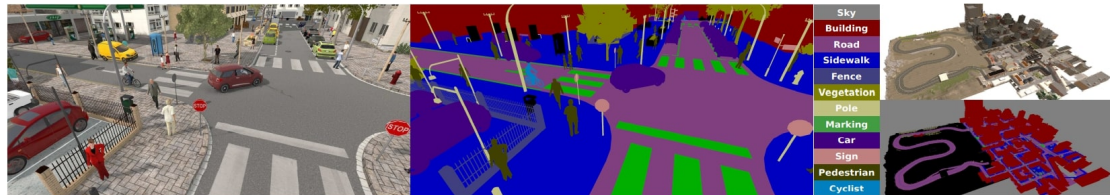


Figure 2.10: The SYNTHIA Dataset. A sample frame (left) with its semantic labels (center) and a general view of the city (right) [60].

#### Synthinel-1

The Synthinel-1 dataset features synthetic data showing building footprints with segmentation masks. The generation of Synthinel-1 is based on CityEngine, which is a tool for efficiently generating large-scale virtual worlds. 2,108 synthetic images with corresponding ground truth are extracted by Kong et al. [62]. Each synthetic images is 572 x 572 pixels in size with a resolution 0.3m/pixel [62]. All ground truth object classes are treated as one single background class, except for the building class. There are nine building styles identified in Synthinel-1 (Figure 2.11).

#### VALID

Chen et al. [63] build a variety of realistic environment in Unreal Engine 4 and create a virtual aerial benchmark suite to enable access to more abundant annotations. A comprehensive Virtual Aerial Image Dataset (VALID) is collected, which consists of

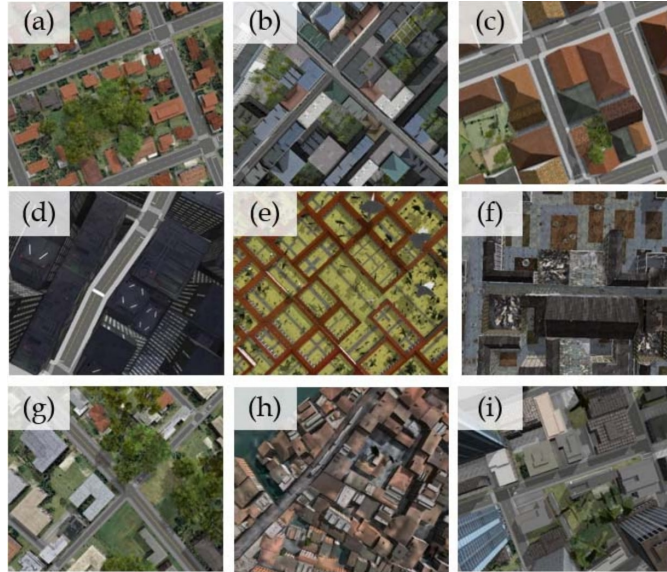


Figure 2.11: An illustration of the nine different virtual city styles used in Synthinel-1. (a) Red roof style; (b) Paris' building style; (c) ancient building style; (d) sci-fi city style; (e) Chinese palace style; (f) Damaged city style; (g) Austin city style; (h) Venice style; (i) modern city style. [62]

6690 high-resolution images with a resolution of  $1024 \times 1024$  [63]. The images are captured in six virtual scenes with five different ambient conditions: sunny, dusk, night, snow and fog as illustrated in Figure 2.12. The VALID dataset labels up to 30 categories, such as tree, plant, road, pavement, land, water, ice, vehicle, building, etc. [63].

## 2.6 Virtual scene and drone/car simulator

Generally, free semantic labels such as segmentation masks, depth images, optic flows, and surface normal images can be rendered by game engine or generated by simulators [64].

### 2.6.1 Simulation platforms

Given models of various objects and layouts of environments, game engines are able to render realistic images and provide large-scale datasets with negligible cost.





Figure 2.12: Six virtual scenes in VALID dataset. There are neighborhood, downtown, airport, night street, snow mountain, and seaside town. [63]

### Gazebo

Gazebo [65] has been one of the most popular simulation platforms used in robotics and related research area. It has a modular design that allows to use different physics engines, sensor models and to create 3D worlds. Gazebo goes beyond monolithic rigid body vehicles and can be used to simulate more general robots. While Gazebo is fairly feature rich it has been difficult to create large scale complex visually rich environments that are closer to the real world.

### The Unity Platform

Unity is a real-time 3D development platform that consists of a rendering and physics engine as well as a graphical user interface called the Unity Editor [66]. Unity is focused on developing a general-purpose engine to support a variety of platforms, developer experience levels, and game types makes the Unity engine an ideal candidate simulation platform for AI research [66]. The flexibility of the underlying engine enables the creation of tasks ranging from simple 2D gridworld problems to complex 3D strategy games, physics-based puzzle, or multi-agent competitive games possible [66]. Furthermore, the Unity Editor enables rapid prototyping and development of games and simulated environments [66].

### Unreal Engine

*Unreal Engine* [67] enables game developers and creators to realize real-time 3D content and experiences with greater fidelity and flexibility. It brings some of the cutting

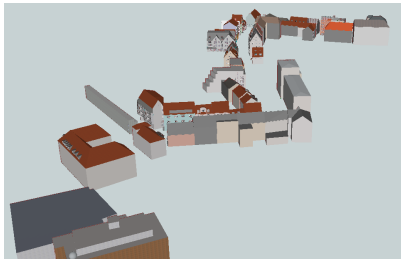
edge graphics features such as physically based materials, photometric lights, planar reflections, ray traced distance field shadows, lit translucency etc [11].

### 2.6.2 Available 3D city models for simulation

3D city models are required as input in simulation platforms for providing semantic information.

#### LoD3 Road Space Models

LoD3 Road Space Models [68] are contained in CityGML [69] dataset in the area of Ingostadt (Figure 2.13). The dataset is manually modeled based on the mobile laser scannings (MLS) with relative accuracy in the range of 1-3cm. A complementary OpenDRIVE [70] dataset is also available, which includes the road network, traffic lights, fences, vegetation and so on.



(a) Overview of LoD3 models.



(b) LoD3 models from a driving view.

Figure 2.13: Illustration of LoD3 models. [68]

#### Berlin 3D

Wu et al. [71] simulated UAV images from Google Earth Studio and generated ground-truth annotations from an open CityGML model [69]. 15 scenes distributed over the area of Berlin are randomly sampled, which show diversity in building styles and density (Figure 2.14) [71]. In total, 144,000 images together with ground-truth geometry (i.e., depth, surface model, and edges), and semantics are generated.

#### City Park Environment Collection

City Park Environment Collection [72] is a large rectangular virtual world available in Unreal Engine Marketplace [73]. The model includes a demo scene of a city park of

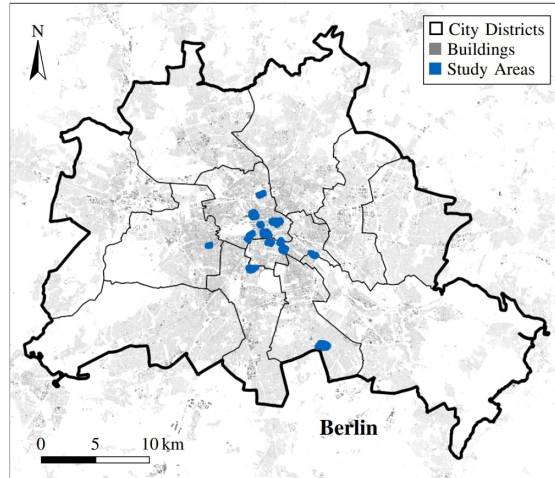


Figure 2.14: 15 areas distributed over the city of Berlin. [71]

over 200 acres. In the scene, there are playgrounds, green areas, lakes, cafes, roads, etc. (Figure 2.15).



Figure 2.15: A snapshot of the city park 3d model. [72]

### 2.6.3 Drone/car simulator

Open source drone/car simulators are widely available, such as CARLA [9], DeepDrive, AirSim [11], etc. These simulators have been used to generate large-scale synthetic datasets with high-level semantic labels including depth, contours, surface normal, segmentation mask, and optical flow for training deep networks [64].

### CARLA

CARLA (CAR Learning to Act) [9] is an open-source driving simulator implemented using Unreal Engine 4. It contains two professionally designed towns with buildings, vegetation, and traffic signs, as well as vehicular and pedestrian traffic. A wide range of environmental conditions can be specified, including weather and time of day. A number of such environmental conditions are illustrated in Figure 2.16.



Figure 2.16: Simulated urban environments [9].

### DeepDrive

DeepDrive [74] is a simulator designed for training self-driving AI models, developed as an Unreal Engine plugin. It provides 8 RGB cameras with  $512 \times 512$  resolution at close to real-time rates (20Hz), as well as a generated 8.2-hour video dataset. An example of the camera viewpoint when simulating driving in DeepDrive is shown in Figure 2.17



Figure 2.17: A snapshot from DeepDrive

### AirSim

AirSim [11] can simulate both driving and flying rendered with Unreal Engine 4. AirSim provide images data generators for RGB images, depth maps, semantic and instance segmentations (Figure 2.18). The images can be generated by simulating a car drive or a drone flight in bird's eye perspective in Unreal simulator. Data such as image size, camera parameters (i.e., exposure, field of view), camera position and orientation can be configured.



Figure 2.18: A snapshot from AirSim shows an aerial vehicle flying in an urban environment [11].

### 3 Methodology

In our study, synthetic data is generated based on the Unreal Engine v4.27 [10] and AirSim [11]. Unreal Python API [75] and the associated Python Script Plugin [76] are used to script to automate the Unreal Editor using Python. Therefore, it is possible to construct large-scale asset management pipelines or workflow for the purpose of our study.

This chapter presents an approach that allows generating synthetic datasets in a simulated environment to support computer vision tasks (Figure 3.1). In section 3.2, vehicle assets are placed in the virtual world with certain constraints. Then segmented aerial images of the vehicles are generated in section 3.3. The next step is to create bounding boxes of the segmented images, and then feed the annotation information to ReDet detector to detect vehicles, which is described in section 3.4.

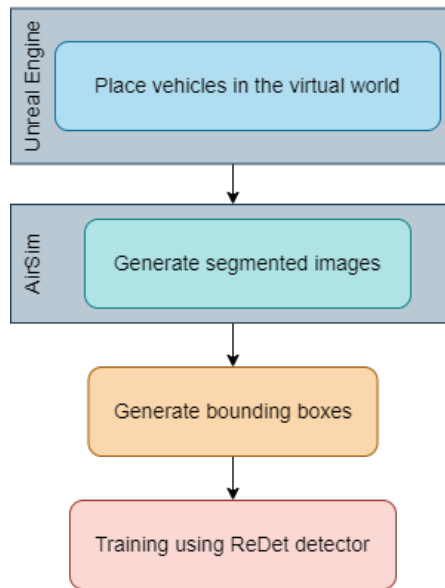


Figure 3.1: Overview of the method

Virtual scenes are artificial scenes created from game engines to simulate the real world with a high level of fidelity. The virtual scenes created in Unreal Engine can be

filled with objects that are downloaded from Unreal Engine marketplace. The number of objects placed and the specific area to be placed on in the virtual world can be configured at the beginning of the data generation. In this study, the object to be placed in the virtual scene is vehicle and the specific area to be placed on is road.

## 3.1 Vehicle assets

The vehicle assets used in this study are *Vehicle Variety Pack* [77] and *Vehicle Variety Pack Volume 2* [78] (Figure 3.2), which are downloaded directly from the Unreal Engine marketplace. *Vehicle Variety Pack* includes five unique vehicles: sport car, hatchback, pickup-truck, SUV, and box truck. *Vehicle Variety Pack Volume 2* includes four models of vehicles: Sedan, box truck, SUV and campervan. All vehicles are designed to have enough details such as the body, undercarriage, interior and window.

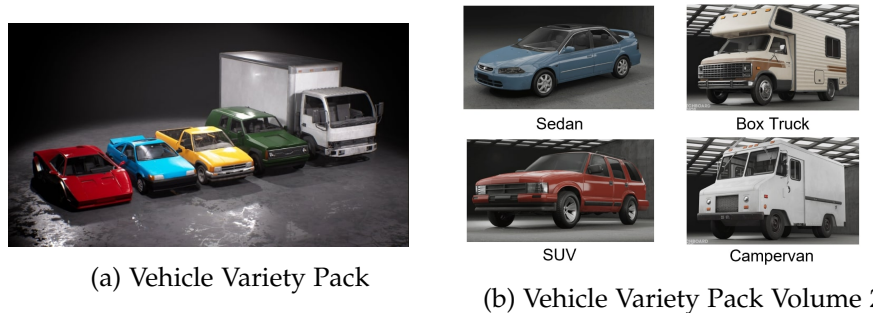


Figure 3.2: Vehicle assets used in the study.

### 3.1.1 Creating vehicles with various colors

For the purpose of increasing the diversity of the vehicle colors, it is desired to increase the number of colors for each vehicle type. For each vehicle color, a new *Dynamic Material Instance* must be created. In the *blueprint* [79] of each vehicle instance, the original color is desaturated and a new customized color is parameterized and applied to the vehicle. Commonly added colors are white, grey, black, blue, and silver. In the end, the vehicle assets expand from nine vehicles to approximately 50 vehicles of nine models but various colours.

## 3.2 Vehicle placement

It is part of the thesis objectives to create realistic scenarios in the virtual world for vehicle placement. Since the semantic information of the position and orientation of the road in City Park Environment Collection is missing, it is necessary to create constraints for the realistic vehicle placement approach. The corresponding constraints are listed as follows:

- Vehicles must be placed only on roads
- Vehicles must be placed with a distance away from the edge of the road
- Vehicles must be oriented as the road direction
- Vehicles must not overlap each other nor any other objects in the scene

The workflow of vehicle placement is illustrated in Figure 3.3.

### 3.2.1 Obtain initial vehicle position

The challenge for obtaining the vehicle position under the constraints is that the semantic information for road in the virtual world is missing. The road system is merged as one road *Actor* in City Park Environment Collection [72]. In this case, the position of the road cannot be simply obtained via accessing the bounding box. This is because each bounding box provides four points with the coordinates of  $X_{max}$ ,  $Y_{max}$ ,  $X_{min}$  and  $Y_{min}$  of the merged road segments. However, what is required is the points that enclose the shape of the road. Due to this situation, alternative approach with the use of *line trace* [80] is used to extract semantic information of the roads.

#### line trace

The line trace offers a method for getting feedback on what is present along a line segment [80]. It is used by providing two end points (Start and end locations) and the physics system "traces" a line segment between those points, reporting any *Actors* (with collision) that it hits (Figure 3.4). When tracing, it returns the first *Object* hit by the trace. Then we can retrieve the *Object Label* to identify whether the hit object is road.

The line traces are generated randomly in the virtual scene. When the hit *Object* is road, we keep the hit location for further computation; when the hit *Object* is not road, we ignore the location Figure 3.5.



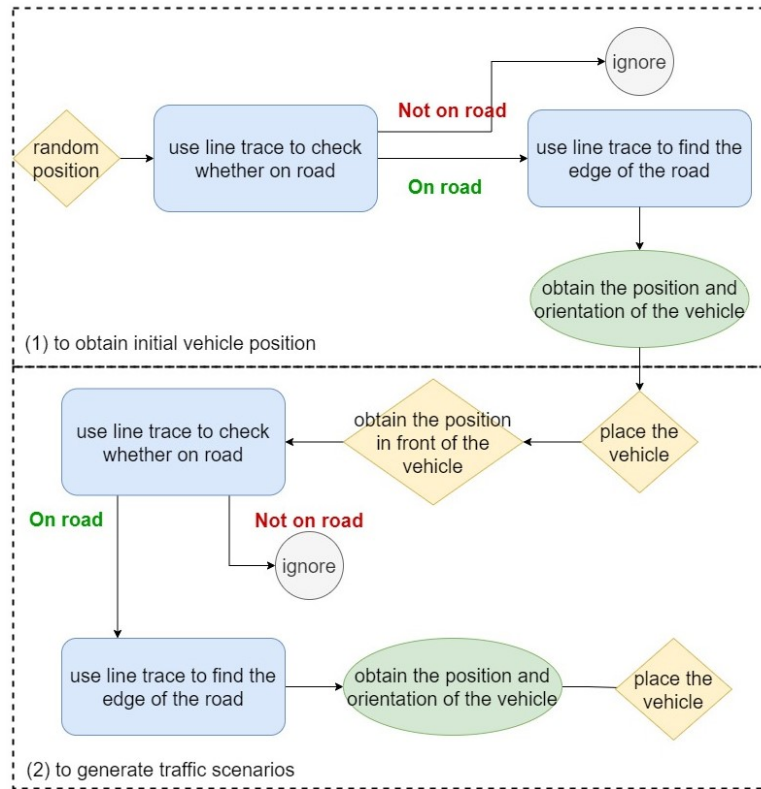


Figure 3.3: Workflow of vehicle placement method. Step (1) is to obtain initial vehicle position; step (2) is to be repeated to generate traffic scenarios.

### 3.2.2 Extract orientation information

The next step is to find the closest edge of the road and obtain the position and orientation of the vehicle. The position of the vehicle must be at a certain distance away from the edge of the road.

As discussed in the previous section, a line trace is generated at a random location in the virtual scene. After identify the Object hit by the trace as road, a second line trace is generated with 0.05 m apart. If the Object hit by the second line trace is also road, a third line trace is generated in the same direction with 0.05 m apart. This process repeats until the Object hit by the  $N^{th}$  line trace is no longer road, the number of line traces that hit the road is recorded for this direction. The same process is repeated for every 10 degrees in a circle centered by the initial random location.

Afterwards, the number of line traces that hit the road at every 10 degrees centered

### 3.2. VEHICLE PLACEMENT

---

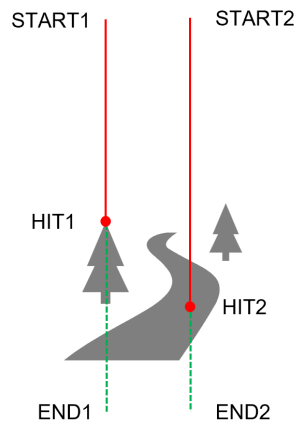
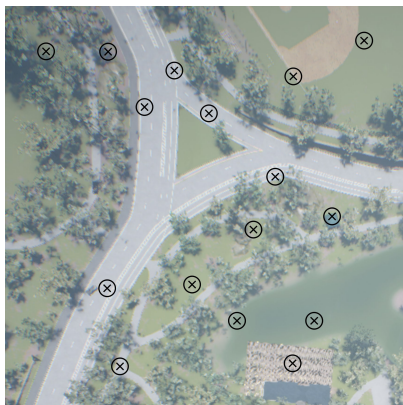
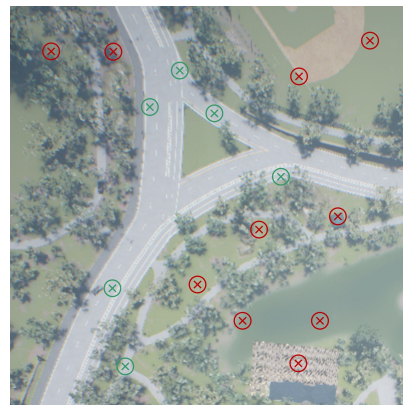


Figure 3.4: Illustration of how line trace works. The line traces start above the landscape (START1, START2) and trace a vertical line down to beneath the landscape (END1, END2). The first Object that the line trace encounters will be returned (HIT1, HIT2). In this diagram, line trace 1 hits a tree; line trace 2 hits road.



(a) The hit locations on the landscape are randomly distributed and indicated by black circles.



(b) The locations indicated by green circles are kept (on road); the ones indicated by red circles are ignored.

Figure 3.5: Example of how line trace helps to identify road Objects.

by the initial random location is recorded together with the direction on a list. The direction with the minimum number of line traces that hit the road is the direction perpendicular to the road edge (Figure 3.6), based on which information, the orientation

of the road can be computed. Along that direction, the last point that is not on road is identified as the edge of the road. A distance  $d$  is configured to place the vehicle away from the road edge and the orientation of the vehicle is configured to be the same as the orientation of the road. Figure 3.7 illustrated the approach in detail.

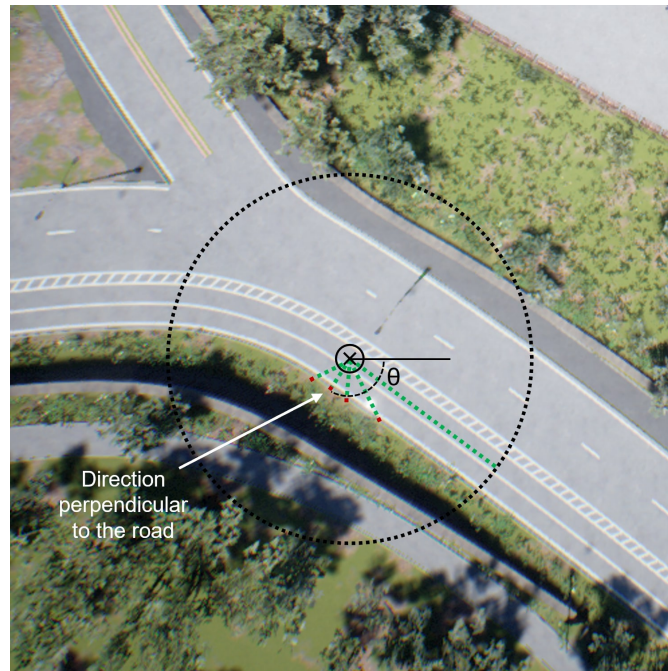


Figure 3.6: Approach to find the orientation of the road. Green dots indicate the hit location on the road, red dots indicates the hit location outside of the road border. The number of green dots is counted along each radius line.

#### Actors to ignore

There are some exceptional scenarios that need to be considered. Figure 3.8a shows two examples when the current criteria for the line trace method would fail. In the scenario when there is a tree growing near the road. The tree crown might cover the road partially in bird's eye view (Figure 3.8b). In this case, the line trace would return the tree as the first Object encountered (HIT1 in Figure 3.8a), yet the location should still be regarded as road. We need to configure additional criteria for the line trace method in order to obtain the correct result.

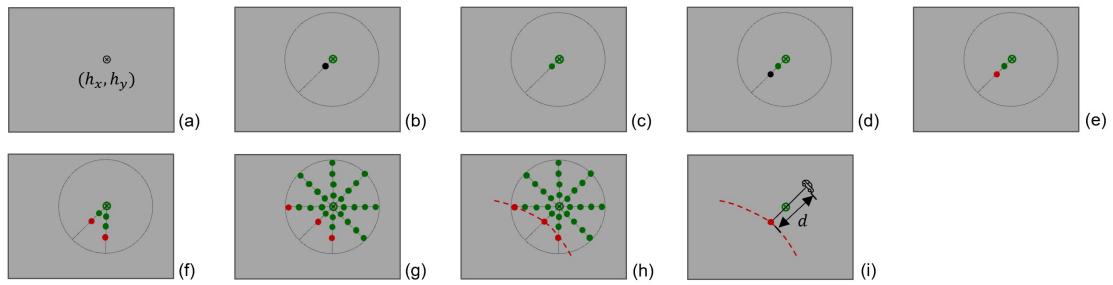
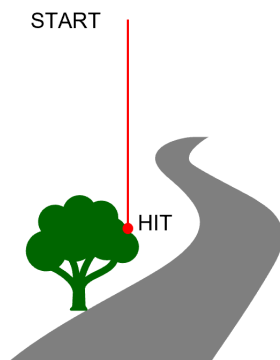


Figure 3.7: Illustration of the approach to find the edge of the road. Green dots indicate on road and red dots indicate not on road. (a) a random point hit by line trace; (b) a second point hit by a line trace with 0.05 m apart; (c) the second point is on road; (d) a third point hit by a line trace along the same direction; (e) the third point is not on road; (f) the same process is repeated along another direction; (g) the same process is repeated for a whole circle; (h) the edge of the road is known based on the break points; (i) a vehicle is placed based on the road edge information.



(a) Graphical illustration.



(b) Example in the virtual scene.

Figure 3.8: Illustration of the exceptional scenarios.

### Avoiding overlaps

The vehicle location is recorded every time after the vehicle is placed in the virtual scene. In order to avoid vehicles overlapping each other, before placing a new vehicle, it is checked that the new location is at a distance (i.e., 8 m) away from the rest of the

vehicles.

### 3.2.3 Traffic scene generation

In order to generate a more realistic virtual world for vehicle detection, traffic scenarios must be taken into consideration. Figure 3.9 illustrates three different traffic scenarios generated in the virtual world.

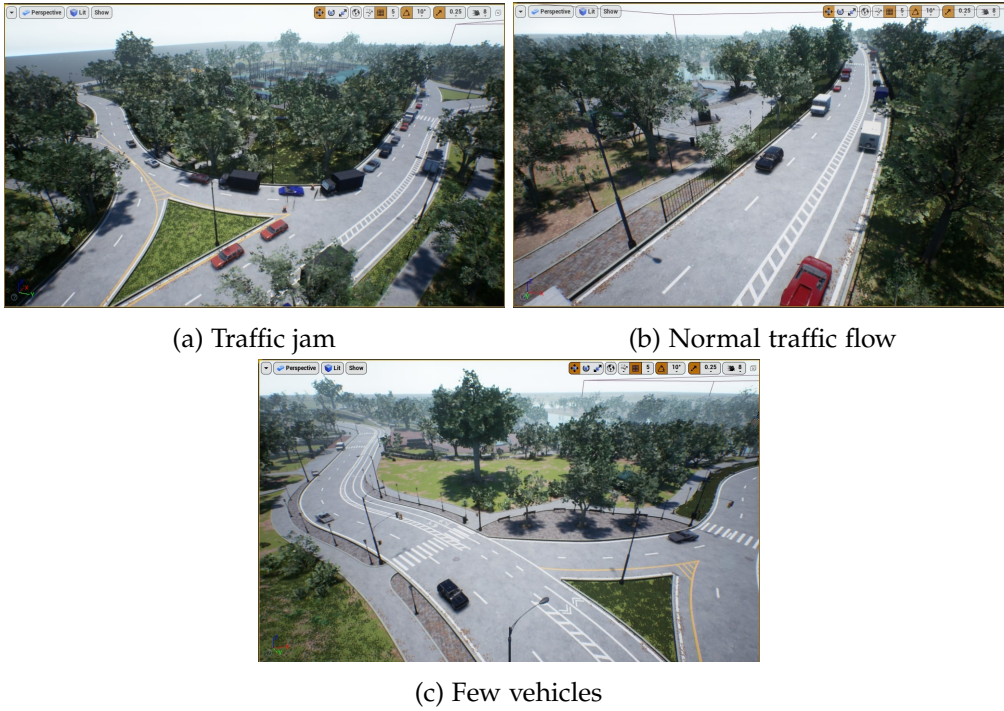


Figure 3.9: Illustration of the three traffic scenarios.

Following the previous section, with known position  $(h_x, h_y)$  and orientation  $\theta$  of the placed vehicle, the next possible position for vehicle placement is computed. If the distance in between vehicles is defined as  $s$ , the next possible position  $(h_x^1, h_y^1)$  is computed as:

$$h_x^1 = h_x \pm abs(s * \cos(\theta)) \quad (3.1)$$

$$h_y^1 = h_y \pm abs(s * \sin(\theta)) \quad (3.2)$$

Since all the vehicles are placed to the right side of the road, the plus or minus sign depends on the orientation  $\theta$ . Figure 3.10 illustrates this step.

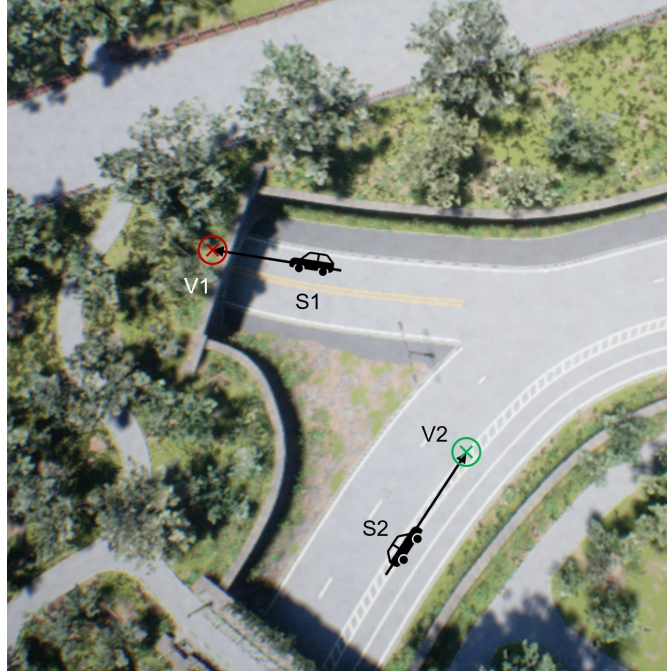


Figure 3.10: Step of finding the possible position for vehicle placement. S1 and S2 are two examples of the initially placed vehicles. V1 and V2 are the two possible locations in the front. Red indicates the new position is not on the road; green indicated the new position is on the road.

After acquiring the position in front of the placed vehicle  $(h_x^1, h_y^1)$ , a line trace is generated again to check whether the position is on the road. In the case when the position is on the road, a second line trace is generated with 0.05 m apart. This step is the same as the step introduced in subsection 3.2.2, yet instead of generating line traces every 10 degrees, now only line traces along two directions are generated. One direction is the direction perpendicular to the orientation of the placed vehicle. The other direction is 10 degrees apart. Since prior knowledge of the road orientation is known, with only two directions, the road orientation next to the new position can be obtained. Figure 3.11 illustrates this step.

The two red points in Figure 3.11 indicates the edge of the road, when connected, the direction is the orientation of the road. A distance  $d$  away from the center of the two points is the location of the second vehicle. Figure 3.12 illustrates the traffic generation approach step by step. These steps are repeated until either the new position is not on the road, or the number of the placed vehicle along the traffic line reaches the configured number.

### 3.2. VEHICLE PLACEMENT



Figure 3.11: Step of finding the orientation at new location V2. Green dots indicates that the line trace encounters road; red crosses indicates the line trace doesn't encounter road.

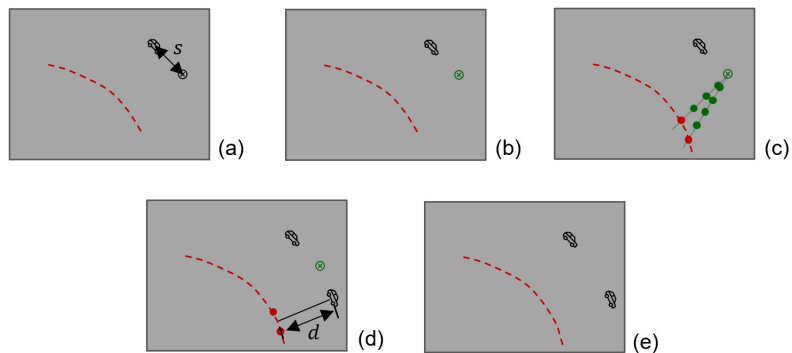


Figure 3.12: Illustration of the traffic generation approach. Green dots indicate on road; red dots indicate not on road. (a) the new position with a distance  $s$  in front of the placed vehicle; (b) the new position is on the road; (c) find the edge of the road; (d) compute the new position and orientation; (e) place the new vehicle

### Double lanes

One further implementation of creating a realistic virtual scene is to generate two lanes of vehicles driving in the opposite directions. Since a single traffic line is created and the position of each placed vehicle is recorded, a random vehicles from the list is selected and rotated by 180 degrees. With known position and orientation, the steps of (c)-(e) from Figure 3.12 are repeated to place a new vehicle of the opposite lane. An example of the generated scene is shown in Figure 3.9b.

### Parameters

Figure 3.13 shows the parameters used to generate the distribution of distance in between vehicles. The distance is measured from the center of each vehicle. For each traffic scenario, skewed and normal distributions are used. The parameters for each scenario is shown in Table 3.1.

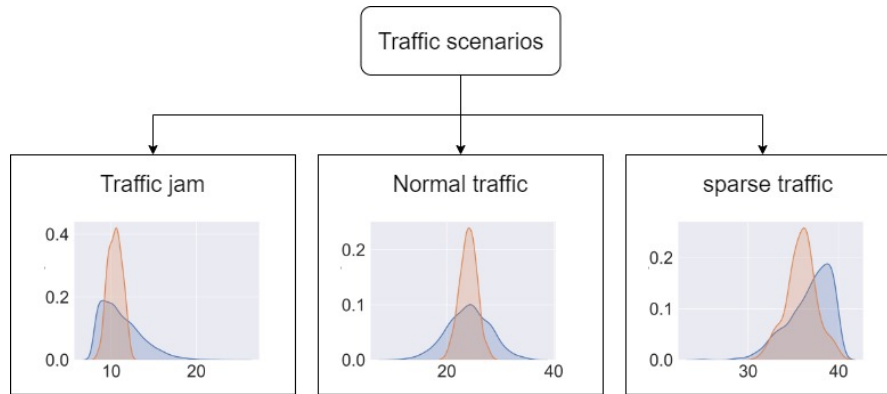


Figure 3.13: Three traffic scenarios and their corresponding distribution of the distance in between vehicles.

For traffic jam scenario,  $F^1$ , it consists of five parameters that define the structure:

$$F^1 = \{skewness^1, \mu^1, \sigma^1, d^1, d_{min}^1, d_{max}^1, N^1\} \quad (3.3)$$

The mode of distance in between vehicles is between 8 m to 10 m. The reason why the distance can not be shorter than 8 m is because that is the minimum space needed when there are two vans placed in a row. Normal traffic flow,  $F^2$ , consists of four parameters:

$$F^2 = \{skewness^2, \mu^2, \sigma^2, d_{min}^2, d_{max}^2, N^2\} \quad (3.4)$$



### 3.3. SEGMENTED IMAGES GENERATION

Parameters	Traffic jam	Normal traffic	Sparse traffic
skewness	30	0	-30
mean ( $\mu$ )	10.4 m	24 m	36 m
standard deviation ( $\sigma$ )	0.8 m	1.6 m	1.6 m
fixed distance ( $d$ )	8 m	-	-
minimum distance ( $d_{min}$ )	7.5 m	11.5 m	25 m
maximum distance ( $d_{max}$ )	23.5 m	37.9 m	40 m
number of vehicles ( $N$ )	$\mu = 30m; \sigma = 8m$		

Table 3.1: Parameters used for traffic scenario generation.

The mean distance in between vehicles is set to 24 m. Sparsely placed vehicles,  $F^3$ , consists of four parameters:

$$F^3 = \{skewness^3, \mu^3, \sigma^3, d_{min}^3, d_{max}^3, N^3\} \quad (3.5)$$

The mode of distance in between vehicles is 36 m. For each scenario, the number of the vehicles along a traffic line is selected from a normal distribution (Table 3.1).

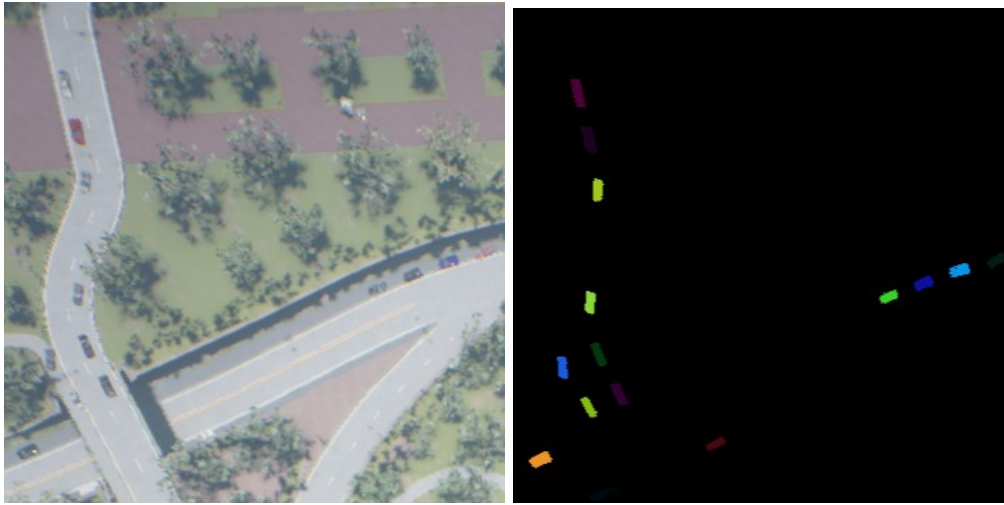
## 3.3 Segmented images generation

### 3.3.1 AirSim set up

"Computer Vision" is an AirSim simulation mode, which is selected so that there is no vehicles and physics to simulate, and cameras can be positioned in any arbitrary pose to collect images. The position  $(x, y, z)$  of the vehicles in the virtual world coordinates is extracted from Unreal Engine. With known vehicle positions, the positions of the camera is configured accordingly, so that the generated images are guaranteed to include vehicles. To ensure a similar ground sampling distance (GSD) as the EAGLE dataset (12 cm), an altitude of 60 m is configured based on the size of the images (i.e., 1024 x 1024px). The camera angle and twist vary randomly between  $-15^\circ$  and  $15^\circ$  and between  $0^\circ$  and  $360^\circ$  respectively. All images generated are under the same sunny weather condition.

### 3.3.2 Instance segmentation

When generating the ground truth segmentation of the scene, the parameter *ImageType* is specified as *Segmentation* in *ImageRequest*. AirSim assigns value 0 to 255 to each



(a) Scene to be segmented (zoomed in)      (b) Instance segmentation (zoomed in)

Figure 3.14: Illustration of the instance segmentation.

*mesh* available in the environment. This value is then mapped to a specific color in the pallet defined by AirSim. In another word, each value from 0 to 255 has corresponding different RGB values.

For each vehicle captured in the segmentation image, a different color is assigned (Figure 3.14). The reason for this is to distinguish vehicles close to each other when creating bounding boxes.

### 3.4 Bounding boxes generation

Each RGB color (except the background color (0,0,0)) in the segmented image is extracted individually in order to generate the bounding box (Figure 3.15).

Then a dilation morphological transformation is applied with a kernel size of (3, 3). Afterwards, each extracted RGB image is converted to grey scale to find the contour of the vehicle segmentation. In case of occlusion (Figure 3.16), the contour of one vehicle segmentation can return multiple regions. The region with the largest area is used as the basis for the bounding box, because it is the most identifiable part. Each boxing box records four values: the center of the width in x coordinate ( $cx$ ), the center of the height in y coordinate ( $cy$ ), the length of the width ( $w$ ), and the length of the height ( $h$ ). The computation of each value is shown in the following equations, based on the coordinates of a bounding box in Figure 3.17.



Figure 3.15: An illustration of extracting individual RGB color for the preparation of bounding box generation.



(a) A scene of an occluded vehicle.



(b) The segmentation of the occluded vehicle.

Figure 3.16: An illustration of occluded vehicles.

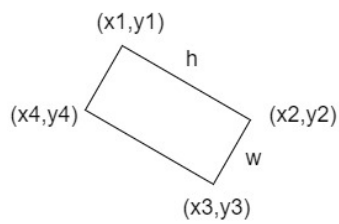


Figure 3.17: The four coordinates of a bounding box.

$$cx = (x_{min} + x_{max})/2 \quad (3.6)$$

$$cy = (y_{min} + y_{max})/2 \quad (3.7)$$

$$w = \text{sqrt}((x1 - x2)^2 + (y1 - y2)^2) \quad (3.8)$$

$$h = \text{sqrt}((x3 - x2)^2 + (y3 - y2)^2) \quad (3.9)$$

### 3.5 Training using ReDet detector

For training the detector, transfer learning technique is used in this thesis. Transfer learning is a powerful deep learning technique in which pre-trained models can be used for feature extraction and fine tuning. This technique can be used in object detection. The advantage of using this technique is that it saves time to train the network from the start and less data is required [81]. The training can also be done using a central processing unit (CPU) even without the computational power of graphics processing unit (GPU). Instead of creating new model from-scratch, pre-trained models can be used. ReDet [82] is one of these models.

#### 3.5.1 ReDet detector

The object detector used in our study is a *Rotation-equivariant Detector (ReDet)*, because it models the orientation variation. ReDet is a Faster R-CNN, which consists of two parts, the rotation equivariant feature extraction and the rotation invariant feature extraction (Figure 3.18) [82].

The first stage is to adopt the rotation-equivariant backbone to extract rotation-equivariant features, followed by RPN and RoI Transformer (RT) to generate Rotated RoIs (RRoIs) (Figure 3.18) [82]. In the second stage in Figure 3.18, a Rotation-invariant RoI Align (RiRoI Align) is used [82]. RiRoI Align includes both spatial alignment and orientation alignment. Its task is to transform the rotation equivariant features to obtain rotation-invariant features (instance level). The so-called rotation-invariant means that, no matter how the input changes (rotation), the output is always the same. RiRoI Align generates rotation invariant features for RoI-wise classification and bounding box (bbox) regression [82].

Unlike most general object detectors that uses Horizontal Bounding Boxes (HBBs) [22, 83], ReDet locate and classify objects with Oriented Bounding Boxes (OBBs), which

### 3.5. TRAINING USING REDET DETECTOR

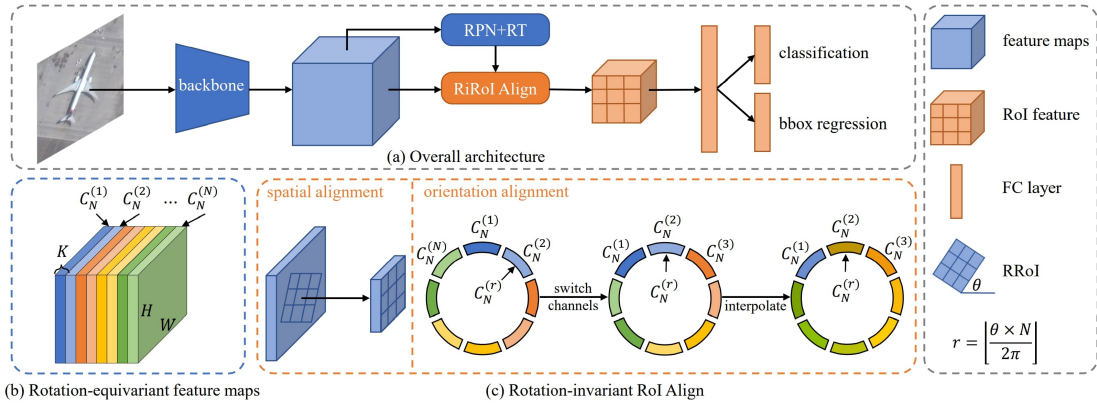


Figure 3.18: Overview of ReDet. (a) Overall architecture of ReDet. (b) Rotation-equivariant feature maps. (c) Rotation-invariant RoI Align. [82]

provide more accurate orientation information of objects [82]. ReDet can accurately predict the orientation and reduce the complexity of modeling orientation variations [82].

## 4 Evaluation and Analysis

This chapter evaluates the vehicle detection results from nine different experiments (EXPs) using a collection of diverse datasets. The main goal is to analyze the performance of vehicle detection using a larger amount of synthetic data and a limited amount of real-world data.

The results of the experiments are reported in terms of True Positive Rate (TPR), False Negative rate (FNR), Precision Recall curve, and Average Precision (AP). The experimental analysis is performed in nine categories (Table 4.2). There are three different groups of training sets: real-world images, synthetic images, and hybrid images with different proportions of each. There are two sets of testing data: synthetic data from aerial view with the realistic scenarios and real-world data taken with aircraft.

### 4.1 Experiment set up

#### 4.1.1 Synthetic datasets

There are two synthetic datasets used for the thesis, in order to compare the detection performance when trained with the same and different vehicle placement. One is the synthetic dataset with aligned vehicles, as described in chapter 3, The other is the synthetic dataset with randomly oriented vehicles. It is ensured that the number of vehicles in each image is approximately the same as which in the images with aligned vehicles. The number of annotation for both training datasets is listed in Table 4.3. An example of the randomly oriented vehicles in the scene is shown in Figure 4.1.

#### 4.1.2 EAGLE dataset used for training and testing

Half of the EAGLE dataset is used for training; 8 original sized images (5616 × 3722px) are selected for testing, based on their GSD and the similarity of the scene to the virtual world. The GSD for the majority of the EAGLE images is around 12 cm, according to which the GSD of the synthetic images is configured. For the scene selection, images with more green area, such as in the countryside regions, are selected. A comparison of the EAGLE image and the synthetic image for both zoom in and zoom out view is shown in Figure 4.2.



(a) Scene of randomly placed vehicles.

(b) Scene of aligned vehicles.

Figure 4.1: Comparison of scenes with randomly placed vehicles and aligned vehicles.

### 4.1.3 Training with ReDet

The parameters used for training is configured as in Table 4.1. Stochastic Gradient Descent (SGD) is used as the optimization algorithm.

### 4.1.4 Experiments design

There are in total nine categories of experiments designed (Table 4.2). The sample size of each training and testing dataset is listed in Table 4.3. Image size for each dataset is either tiled or configured to be  $1024 \times 1024$ px. The experiments are crafted to evaluate the performance of vehicle detection with our generated synthetic data in three purposes:

1. Evaluation of detection when trained with the same vs. different vehicle placement patterns
2. Evaluation of the domain gap between real and synthetic datasets
3. Evaluation of how much real-world data is needed to have a good performance

4.1. EXPERIMENT SET UP

---



(a) Real-world images from Eagle (tiled)

(b) Synthetic images (zoomed in)



(c) Real-world images from Eagle (zoomed out)

(d) Synthetic images (zoomed out)

Figure 4.2: Visual comparison between real and synthetic images



#### 4.1. EXPERIMENT SET UP

PARAMETER	VALUE
learning rate	0.01
weight decay	0.0001
momentum	0.9
epochs	12
batch size	150
images per batch	7
number of classes	2 (vehicle and non-vehicle)

Table 4.1: Parameters used for traffic scenario generation.

EXPERIMENT	TRAINING DATA	TESTING DATA
EXP1	synthetic images (aligned)	synthetic images (aligned)
EXP2	synthetic images (random)	synthetic images (aligned)
EXP3	synthetic images (aligned)	real-world images
EXP4	real-world images	synthetic images (aligned)
EXP5	synthetic images + 9% real-world images	real-world images
EXP6	synthetic images + 12% real-world images	real-world images
EXP7	synthetic images + 22% real-world images	real-world images
EXP8	synthetic images + 49% real-world images	real-world images
EXP9	real-world images	real-world images

Table 4.2: Training and testing datasets. Ratios of real-world images are based on the number of synthetic images in the dataset.

	Dataset	Images	Annotations
TRAINING DATASET	synthetic (aligned)	419 (1024 x 1024px)	11k
	synthetic (random)	469 (1024 x 1024px)	10k
	real	4k (1024 x 1024px)	100k
TESTING DATASET	synthetic (aligned)	166 (1024 x 1024px)	6k
	real	178 (1024 x 1024px)	5k

Table 4.3: Size of each dataset.

## 4.2 Vehicle detection metrics

It is difficult to evaluate the effectiveness of an object detector without metrics. One metric usually used in the early research of object detection [18] are the "miss rate vs. false positives per-window (FPPW)". However, the FPPW measurement can be flawed and fails to predict full image performance in certain cases [84].

### 4.2.1 True Positive Rate and False Negative Rate

The True Positive Rate (TPR), also known as sensitivity or recall in machine learning, measures the percentage of actual positives that are accurately identified [85]. More information of TPR (i.e., recall) is introduced in the following subsection.

The False Negative Rate (FNR), also known as the miss rate, is the probability that the model incorrectly predicts the negative class [86]. It is calculated as

$$FNR = \frac{FN}{TP + FN} \quad (4.1)$$

FNR ranges from zero to one. The closer the FNR is to zero, the better the prediction.

### 4.2.2 Precision and Recall

Precision is defined as the number of true positives ( $TP$ ) over the number of true positives plus the number of false positives ( $FP$ ).

$$Precision = \frac{TP}{TP + FP} \quad (4.2)$$

Recall is defined as the number of true positives ( $TP$ ) over the number of true positives plus the number of false negatives ( $FN$ ).

$$Recall = \frac{TP}{TP + FN} \quad (4.3)$$

Precision measures how accurate your predictions are, indicating the percentage of correct positive predictions. The best possible precision is one, while the lowest possible is zero. Recall measures how good our detector is at finding all the positives, indicating the percentage of the positive ground-truth objects that our detector finds. The best possible recall is one, while the lowest possible is zero.

### 4.2.3 Precision-Recall curve

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced [87]. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned [87].

In circumstances where recall is critical, decision threshold is an essential parameter playing a major role. A decision threshold is set to 0.5 by default. This means that if the model believes an observation has a 50% or greater chance of being a member of the positive class, it is projected to be a member of the positive class. When the decision threshold is reduced, more true affirmative cases can be caught and precision decreases while recall increases.

The precision-recall curve shows the trade-off between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

### 4.2.4 Intersection over Union (IoU)

To measure the object localization accuracy, the Intersection over Union (IoU) is used to check whether the IoU between the predicted box and the ground truth box is greater than a predefined threshold, by default, 0.5 [3]. If yes, the object will be identified as "successfully detected", otherwise will be identified as "missed". The 0.5-IoU based mAP has then become the de facto metric for object detection problems. The definition is illustrated in Figure 4.3.

### 4.2.5 Average Precision (AP)

In recent years, the most frequently used evaluation for object detection is "Average Precision (AP)". AP summarizes the shape of the precision-recall curve into a single

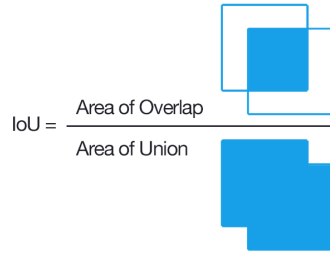


Figure 4.3: Illustration of the definition of IoU [88]

value representing the average of all precisions [89]. AP is defined as the mean precision at a set of eleven equally spaced recall levels  $[0, 0.1, \dots, 1]$ :

$$AP = \frac{1}{11} \sum_{n \in \{0, 0.1, \dots, 1\}} P_{interplated}(r) \quad (4.4)$$

In other words, AP is the weighted sum of precisions at each threshold where the weight is the increase in recall.

### 4.3 Realistic scenario setting

In this section, the effects of vehicle placement in the synthetic training datasets on the model's performance will be evaluated. To compare the realistic scenario setting in two synthetic datasets, one training dataset is images of vehicles in various traffic scenarios, which are lined up and aligned with road directions as described in section 3.2. The other training dataset is images of vehicles placed randomly on roads. In both datasets, the number of samples is approximately 10K. The testing dataset is kept the same, which is synthetic images with aligned vehicle placement.

Figure 4.4 shows the comparison of the corresponding Precision Recall curves. Both experiments are with high precision but low recall. Training with realistic synthetic data has a better detection performance due to the similar pattern of vehicles as in the testing dataset. The visualization of the detection results is shown in Figure 4.6. It can be seen that there are more predicted labels when trained with synthetic dataset consisting aligned vehicles.

The average precision is listed in Table 4.4. With random placed vehicles in training dataset, the detection performance is reduced. Qualitative results of the detector trained on synthetic data are shown in Figure 4.5. Predictions are shown via pink bounding boxes, where the confident score is labeled next to them.

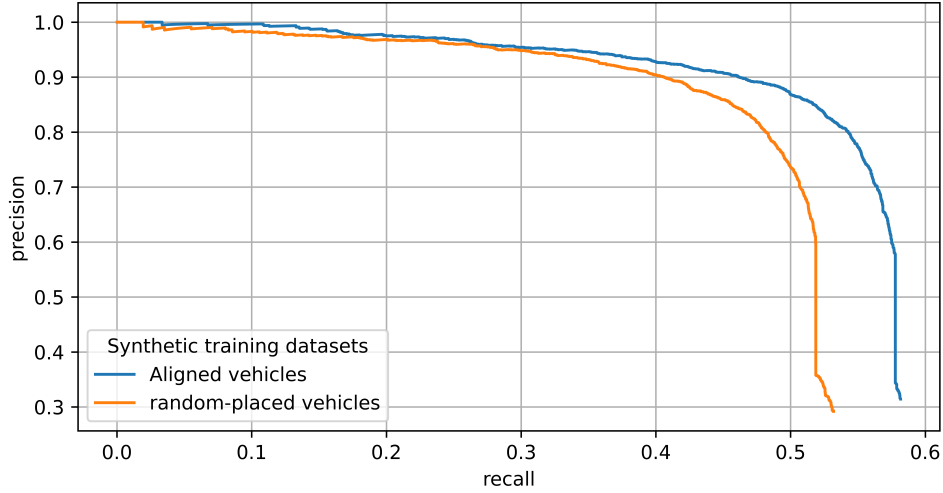


Figure 4.4: Comparison of Precision-Recall curves for training with two synthetic datasets.

EXPERIMENT	TRAINING DATA	TPR	FNR	AP
EXP1	synthetic images (aligned)	58%	42%	54%
EXP2	synthetic images (random)	53%	47%	48%

Table 4.4: Evaluation metrics of experiments 1 (EXP1) and 2 (EXP2).

## 4.4 Domain Gap

Discrepancy between physical simulators and the real world makes it challenging to adapt synthetic data in training. In terms of vehicles in the real world, there can be physical effect variations, such as object texture, traffic scenarios, road construction, weather conditions etc., that creates the reality gap between the real world and physics simulators.

To compare training using synthetic versus real-world data, one experiment is to train with synthetic data and test on real-world data (i.e., EXP3), the other is vice versa (i.e., EXP4), as shown in Table 4.5. For experiment 3, 419 images are used for training and 178 tiled images from the EAGLE dataset are selected for testing, based on their GSD and the similarity of the scene compared with synthetic images. For experiment 4, 159 original-sized images from EAGLE dataset are tiled and used for training. 166



(a) Annotation



(b) Trained with dataset consisting of aligned vehicles.



(c) Trained with dataset consisting of random-placed vehicles.

Figure 4.5: Visualization of the detection performance with different synthetic training datasets



#### 4.4. DOMAIN GAP

synthetic images are used for testing. The whole EAGLE dataset consists of 345 images of  $5616 \times 3744$ px size, and more than 1 million annotated vehicles [5]. The dataset also includes a variety of scenes in aerial photography such as different time, weather, and places ([5]), which means it is challenging for a network trained only on synthetic data to compete. The Precision Recall curves are shown in Figure 4.7.

EXPERIMENT	TRAINING DATA	TESTING DATA	TPR	FNR	AP
EXP3	synthetic images (aligned)	real-world images	12%	88%	5%
EXP4	real-world images	synthetic images (aligned)	44%	56%	24%

Table 4.5: Evaluation metrics of experiments 3 (EXP3) and 4 (EXP4).

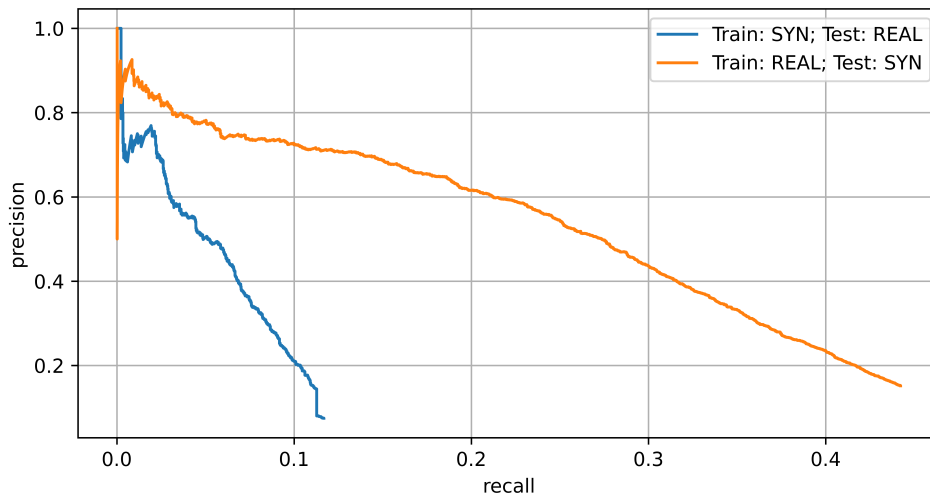


Figure 4.7: Precision Recall curve for Experiments 3 & 4.

Average precision for the network trained on real-world images is higher than the one trained on synthetic images by 0.19. The precision for both experiment results decreases considerably with fluctuations as the threshold of classifier decreases (Figure 4.7). At precision of 1, the threshold is set very high, which makes the results all true positive. Lowering the threshold increases the denominator of precision ( $\frac{T_p}{T_p+F_p}$ ), by increasing the number of results returned, which may lead to an increase in precision. The highest



recall only reaches 0.12 for training with synthetic images and 0.45 for training with real-world images, which is low in both cases.

The visualization of the predicted labels for both experiments is shown in Figure 4.8. Figure 4.8a indicates the true positive and false negative predictions from experiment 3, where most vehicles aligned with the road are detected, yet the ones parked in the parking lot are not detected. Figure 4.8e shows the false positive predictions from experiment 3, where the rooftop air outlets are detected as vehicles. Both false positive and false negative predictions might be due to the fact that the neural network was not trained with rooftop structures and parking lot images. Figure 4.8f shows the predictions from experiment 4. A lot of false negative detection exists in the image, which might be due to the fact that the vehicles in synthetic images are not realistic enough as in the real-world images.

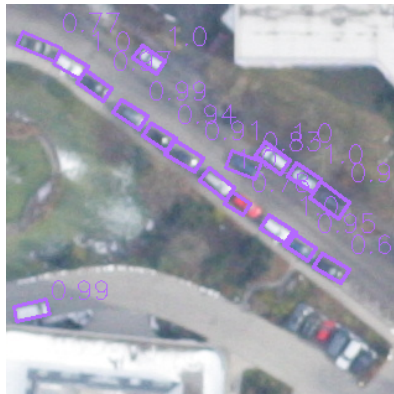
This result indicates that there is a significant domain gap between real and synthetic data, yet training with real-world data outperforms training with synthetic data. The difference in performance can be deduced from the difference in the domain coverage and amount in training data. The real-world data covers a large variety of domains (i.e., time of the day, season of the year) and uses a much larger amount of images (159 images (5616 x 3744px)), yet the synthetic data covers only the same scene and uses 419 images (1024 x 1024px).

## 4.5 Synthetic and real-world data mixing

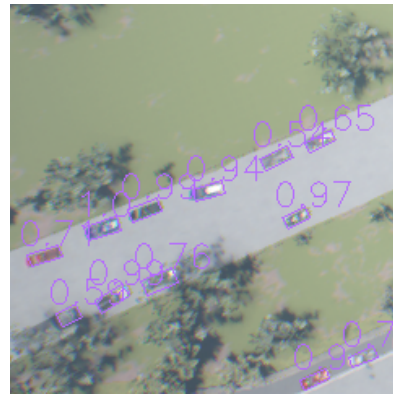
In an attempt to achieve the full real-world data performance with only using a fraction of it, a mixed set of synthetic and real-world data with various ratios are studied. The average precision of each experiment is shown in Table 4.6. The mixed dataset is used in training, while the tests are performed only on real-world data. The synthetic images in experiment 5-8 remain the same both in terms of images themselves and the amount. The variable is the number of real-world images added in the training dataset.

The Precision Recall curve for each experiment is shown in Figure 4.9. As more real-world data are added in the training dataset, both precision and recall increase. For real-world data ratio of 12%, 22%, and 49%, there is a sudden drop of precision at a recall score of 65%. This sudden drop of precision may be because a sudden increase of false positive detection. From Figure 4.8e, we can see that there are a large number of rooftop air outlets in the real-world images. When the detection threshold lowers down, maybe almost all air outlets and similar structure objects are detected as vehicles, which decrease the precision dramatically.

4.5. SYNTHETIC AND REAL-WORLD DATA MIXING



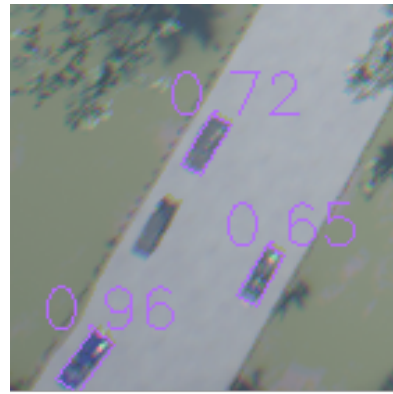
(a) Good detection result of EXP3.



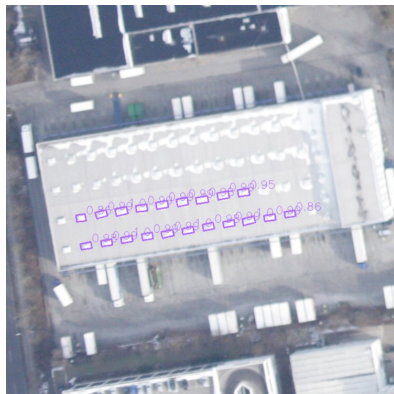
(b) Good detection result of EXP4.



(c) Medium detection result of EXP3.



(d) Medium detection result of EXP4.



(e) Poor detection result of EXP3.



(f) Poor detection result of EXP4.

Figure 4.8: Visualization of the detection performance of experiments 3 and 4.

---

#### 4.5. SYNTHETIC AND REAL-WORLD DATA MIXING

---

The real-world images used in the mixed dataset are selected from EAGLE dataset based on similar GSD and scenes as synthetic images, which are 8 original-sized images in the EAGLE dataset. Compared to the average precision (i.e., 0.77) of training with 22% of the EAGLE dataset, a mixture of 50% synthetic and 50% real-world data achieves a relatively good average precision of 0.63. Since the amount of the training dataset is different, the results of experiment 8 and 9 are not comparable, but can be used for reference. With an increase of 3% of real-world images from experiment 5 to 6, the vehicle detection performance is considerably improved from 0.32 to 0.53 average precision (Figure 4.10). An increase in the real-world data ratio from 9% to 49% results in a logarithmic increase in the performance. With this result, the hypothesis to observe a performance increase by adding a small amount of real-world data compared to the synthetic only training is confirmed.

EXPERIMENT	SYNTHETIC RATIO	REAL RATIO	TPR	FNR	AP
EXP3	100%	0%	12%	88%	5%
EXP5	91%	9%	52%	48%	32%
EXP6	88%	12%	65%	35%	53%
EXP7	78%	22%	69%	31%	60%
EXP8	51%	49%	72%	28%	63%
EXP9	0%	100%	82%	18%	77%

Table 4.6: Evaluation metrics of experiments with mixed synthetic and real-world images for training.

#### 4.5. SYNTHETIC AND REAL-WORLD DATA MIXING

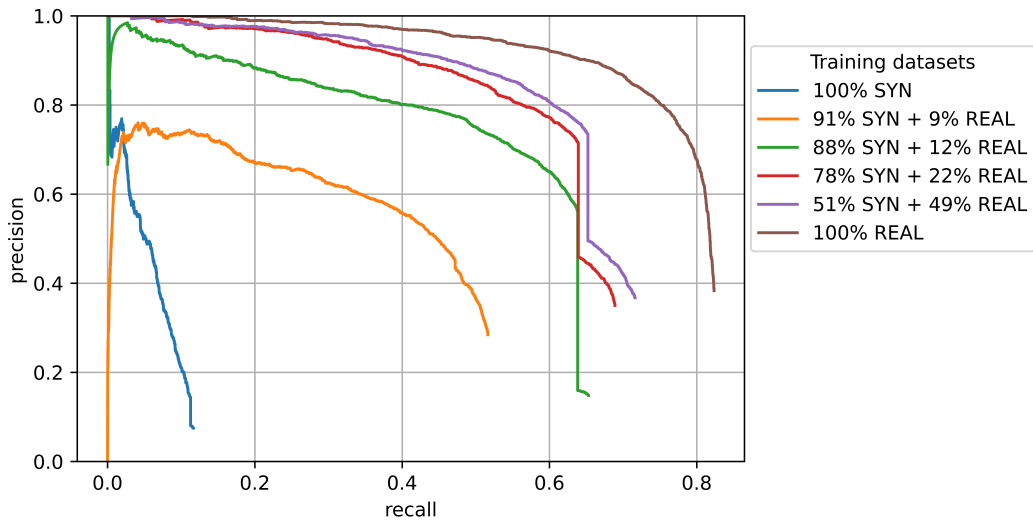


Figure 4.9: Precision Recall curve for Experiments 3 and 6-9, all experiments used real-world data for testing.

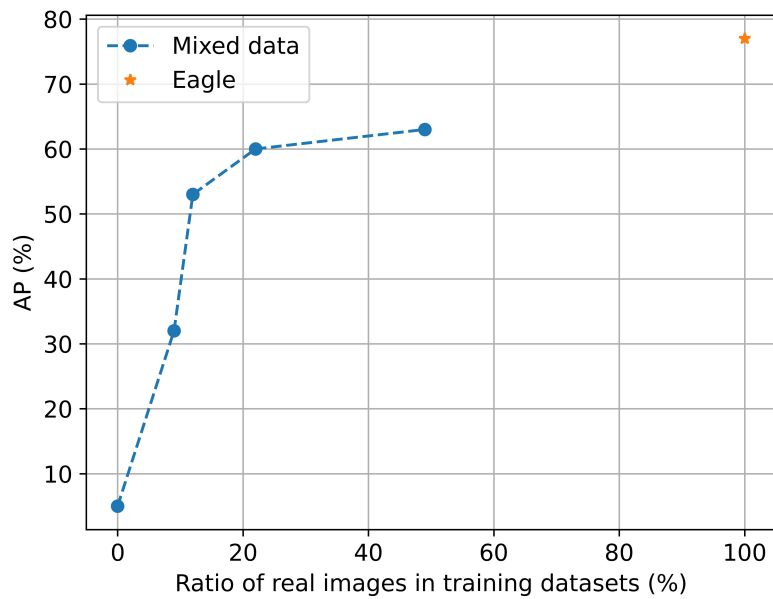


Figure 4.10: Average precision of detection performance with percentage of real-world images in the total training data.

## 5 Discussion

The first task of this thesis is to generate vehicles in a virtual world and to place them with limited semantic information. The constraints for vehicle placement have been satisfied so that various traffic scenarios in the virtual scene are created to imitate the real world.

Since the information such as the width of the road and the position of the road lane markings are missing, based on the approach in section 3.2, the vehicles are not necessarily placed in between the lane markings but rather a fixed distance from the edge of the road. This can be problematic when more complex circumstances are involved, such as when pedestrians and bikers are in the scene. However, the design of this thesis simplifies the scenarios and only focuses on vehicles while lane markings are ignored.

A pipeline for vehicle placement and synthetic image generation using Unreal Python API is created with configured parameters for traffic scenarios and camera settings, etc. The ground truth labeling is automatically generated using AirSim via instance segmentation. In the situation with an inclined camera angle up to 15 degrees, semantic segmentation would incorrectly label the overlapping vehicles in the camera's view as one category, yet instance segmentation separates individual vehicles to avoid the mistake.

There are three goals for the designing of the training experiments. The first is to evaluate the detection performances trained with the same or different vehicle placement patterns. In Table 4.4, the average precision for the detection trained with dataset of randomly placed vehicles (48%) is 6% lower than the detection trained with dataset of aligned vehicles (54%). This difference is mainly due to a lower recall for the former detection. In other words, the neural network trained with randomly placed vehicles has more difficulties to obtain positive predictions. In the visualization of both predictions (Figure 4.5), it can be identified that the detector trained with randomly placed vehicles makes less positive predictions than the other. Therefore, using the same vehicle placement patterns in the training and testing datasets increases the detection performance.

---

The second goal is to evaluate the domain gap between real and synthetic datasets. Training with synthetic images and testing on real-world images (EXP3) reflects how well the real-world images resemble the synthetic images; while training with real-world images and testing on synthetic images (EXP4) reflects the opposite. In Table 4.5, the average precision for EXP3 is much lower than that for EXP4. The very low average precision for EXP3 (5%) might be due to the fact that there are more varieties of vehicles and the background in the real-world images compared to the synthetic images. In other words, the synthetic images have less domain variations than the real-world images, which leads to a large domain gap in between. The average precision for EXP4 is 24%, which also indicates a poor detection performance. By visualizing the predicted labels in Figure 4.8, it can be observed that the color contrast and saturation in the synthetic images are not as strong as in the real-world images. Therefore, one reason for the poor detection performance is that the image qualities of synthetic images, in terms of textures, shapes, colors, etc., do not resemble that of real-world images. Overall, it is indicated that the domain gap between synthetic and real-world images is large.

The third goal is to evaluate the ratio of real-world data needed for having a good detection performance. The experimental results demonstrate that the performance based on a mixture of synthetic and real-world data is comparable to the performance based on only real-world data. The average precision for the experiment trained with only real-world data reaches 77%. Considering the large amount of real-world data from half of EAGLE dataset, experiment 8 trained with half synthetic (51%) and half real-world images (49%) has reached an adequate performance of 63%. It is definitely possible to continue increase the performance by adding more real-world images in the training dataset. However, the trend of the increase of average precision shown in Figure 4.10 indicates a logarithmic growth, which means a large ratio of real-world images are needed to achieve a small increase in the average precision. The concept of these experiments is to use less real-world images to achieve maximum detection performance, because synthetic images can be generated with much less efforts and in great amount, yet real annotated images demands much more time and labour costs. Therefore, continue increasing the ratio of real-world images is not part of the experiment scope. The results of the experiments can be summarized as follows. By adding a small ratio of real-world images in the training dataset, the detection performance improved dramatically, as can be verified in Figure 4.9 and Figure 4.10. As the ratio of real-world images increases, the performance improvement is slowed down and reaches a plateau. Based on the experiments, 20% to 50% of real-world images are able to rise the detection performance to an adequate level (i.e., more than 60%).

To summarize, synthetic data introduces a cost saving measure for data annotation.

---

Training with realistic synthetic data in terms of imitating real world scenarios and optimizing image qualities improves the detection performance. An increase of the real-world image ratio in the training dataset also improves the detection performance dramatically until a certain threshold is reached, where the performance can not be improved even with a great amount of real-world images.

## 6 Conclusions and Outlook

After simulating the virtual world where the vehicles are placed according to the constraints, images can be taken with configured camera settings such as the position and camera parameters. Numerous synthetic images can therefore be generated with accurate ground truth automatically via the generation pipeline presented in the thesis.

It is desired that synthetic images can not only be generated automatically, but also resemble the real world images. For the latter perspective, some further improvements can be investigated. Four factors affect how well synthetic images resemble the real world. One is the quality of the objects in the virtual world in terms of the texture, shape, and color. In this thesis, both the virtual world and the vehicle assets are downloaded directly from Unreal Engine Marketplace. Therefore it is not possible to configure the realism of objects in the virtual world. If a higher quality is desired, virtual worlds from other resources must be investigated.

Another factor is the quality of the images in terms of brightness, contrast and saturation. Compared with the image quality of the real world images, experimental research can be further conducted to acquire the image parameters that generate the most resembling synthetic images.

A third factor is the scenario settings in the virtual world compared with the real world. Different traffic scenarios are generated in this thesis to imitate the real world traffic scenarios. However, there can still be future improvement in the vehicle placement, such as creating parking lot scenarios, dense parking along both sides of the road, driving on multiple lanes, etc.

Last but not the least, the variety of the domains in synthetic images compared to the real world images also affects the resemblance. In this thesis, only one virtual world, City Park Environment Collection, is investigated. In future researches, multiple virtual world with different road situations can be utilized for synthetic images generation, which would increase the domain variety and reduce the domain gap between synthetic and real-world images.



---

When the above factors are considered and investigated, synthetic images for vehicle detection can be improved. In addition, with an increasing ratio of real-world images used in training dataset, the domain gap between training and testing datasets is reduced and therefore, the detection performance increases as shown in Figure 4.9. The amount of synthetic images used for training could also play a role in detection performance. In this thesis, only 419 synthetic images of 1024 x 1024 px are used for training. Further experiments can be done by increasing the number of synthetic images in the training dataset.

Overall, using synthetic images for training is an promising approach for reduction of labour and time-consumption. This thesis provides an approach for automatic synthetic data generation using Unreal Engine and AirSim. The synthetic images generation pipeline can be a guidance or implemented directly for future synthetic image generation. Experiments with different research interests are conducted, which provides an insight of how to investigate the resemblance of synthetic images to real world images. The logarithmic growth of detection performance with an increase of real-world image ratios also provides inspiration for further research on training with mixed synthetic and real-world images.

## List of Figures

2.1	The flowchart of machine learning-based object detection [25]. . . . .	5
2.2	The pipeline of general CNN architecture [35] . . . . .	7
2.3	R-CNN architecture [22]. . . . .	8
2.4	Fast R-CNN architecture [23]. . . . .	9
2.5	Faster R-CNN architecture and RPN [38] . . . . .	9
2.6	The YOLO detection system [40] . . . . .	10
2.7	Images illustrating the different categories of the dataset. From left to right: car, truck, camping car, tractor, plane, boat, other, pickup and van [46]. . . . .	11
2.8	Samples of annotated images in DOTA. There are three samples for each category, except six for <i>large-vehicle</i> . [44] . . . . .	12
2.9	Examples of annotated images [5] . . . . .	13
2.10	The SYNTHIA Dataset. A sample frame (left) with its semantic labels (center) and a general view of the city (right) [60]. . . . .	15
2.11	An illustration of the nine different virtual city styles used in Synthinel-1. (a) Red roof style; (b) Paris' building style; (c) ancient building style; (d) sci-fi city style; (e) Chinese palace style; (f) Damaged city style; (g) Austin city style; (h) Venice style; (i) modern city style. [62] . . . . .	16
2.12	Six virtual scenes in VALID dataset. There are neighborhood, downtown, airport, night street, snow mountain, and seaside town. [63] . . . . .	17
2.13	Illustration of LoD3 models. [68] . . . . .	18
2.14	15 areas distributed over the city of Berlin. [71] . . . . .	19
2.15	A snapshot of the city park 3d model. [72] . . . . .	19
2.16	Simulated urban environments [9]. . . . .	20
2.17	A snapshot from DeepDrive . . . . .	21
2.18	A snapshot from AirSim shows an aerial vehicle flying in an urban environment [11]. . . . .	21
3.1	Overview of the method . . . . .	22
3.2	Vehicle assets used in the study. . . . .	23
3.3	Workflow of vehicle placement method. Step (1) is to obtain initial vehicle position; step (2) is to be repeated to generate traffic scenarios. . . . .	25

3.4	Illustration of how line trace works. The line traces start above the landscape (START1, START2) and trace a vertical line down to beneath the landscape (END1, END2). The first Object that the line trace encounters will be returned (HIT1, HIT2). In this diagram, line trace 1 hits a tree; line trace 2 hits road. . . . .	26
3.5	Example of how line trace helps to identify road Objects. . . . .	26
3.6	Approach to find the orientation of the road. Green dots indicate the hit location on the road, red dots indicates the hit location outside of the road border. The number of green dots is counted along each radius line.	27
3.7	Illustration of the approach to find the edge of the road. Green dots indicate on road and red dots indicate not on road. (a) a random point hit by line trace; (b) a second point hit by a line trace with 0.05 m apart; (c) the second point is on road; (d) a third point hit by a line trace along the same direction; (e) the third point is not on road; (f) the same process is repeated along another direction; (g) the same process is repeated for a whole circle; (h) the edge of the road is known based on the break points; (i) a vehicle is placed based on the road edge information. . . .	28
3.8	Illustration of the exceptional scenarios. . . . .	28
3.9	Illustration of the three traffic scenarios. . . . .	29
3.10	Step of finding the possible position for vehicle placement. S1 and S2 are two examples of the initially placed vehicles. V1 and V2 are the two possible locations in the front. Red indicates the new position is not on the road; green indicated the new position is on the road. . . . .	30
3.11	Step of finding the orientation at new location V2. Green dots indicates that the line trace encounters road; red crosses indicates the line trace doesn't encounter road. . . . .	31
3.12	Illustration of the traffic generation approach. Green dots indicate on road; red dots indicate not on road. (a) the new position with a distance $s$ in front of the placed vehicle; (b) the new position is on the road; (c) find the edge of the road; (d) compute the new position and orientation; (e) place the new vehicle . . . . .	31
3.13	Three traffic scenarios and their corresponding distribution of the distance in between vehicles. . . . .	32
3.14	Illustration of the instance segmentation. . . . .	34
3.15	An illustration of extracting individual RGB color for the preparation of bounding box generation. . . . .	35
3.16	An illustration of occluded vehicles. . . . .	35
3.17	The four coordinates of a bounding box. . . . .	35

3.18	Overview of ReDet. (a) Overall architecture of ReDet. (b) Rotation-equivariant feature maps. (c) Rotation-invariant RoI Align. [82] . . . . .	37
4.1	Comparison of scenes with randomly placed vehicles and aligned vehicles. . . . .	39
4.2	Visual comparison between real and synthetic images . . . . .	40
4.3	Illustration of the definition of IoU [88] . . . . .	44
4.4	Comparison of Precision-Recall curves for training with two synthetic datasets. . . . .	45
4.5	Visualization of the detection performance with different synthetic training datasets . . . . .	46
4.6	Examples of detection results from both EXP1 and EXP2. . . . .	47
4.7	Precision Recall curve for Experiments 3 & 4. . . . .	48
4.8	Visualization of the detection performance of experiments 3 and 4. . . .	50
4.9	Precision Recall curve for Experiments 3 and 6-9, all experiments used real-world data for testing. . . . .	52
4.10	Average precision of detection performance with percentage of real-world images in the total training data. . . . .	52

# List of Tables

3.1	Parameters used for traffic scenario generation. . . . .	33
4.1	Parameters used for traffic scenario generation. . . . .	41
4.2	Training and testing datasets. Ratios of real-world images are based on the number of synthetic images in the dataset. . . . .	41
4.3	Size of each dataset. . . . .	41
4.4	Evaluation metrics of experiments 1 (EXP1) and 2 (EXP2). . . . .	45
4.5	Evaluation metrics of experiments 3 (EXP3) and 4 (EXP4). . . . .	48
4.6	Evaluation metrics of experiments with mixed synthetic and real-world images for training. . . . .	51

## Bibliography

- [1] P. Ongsulee. “Artificial intelligence, machine learning and deep learning.” In: *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*. IEEE. 2017, pp. 1–6.
- [2] Q. Wu, Y. Liu, Q. Li, S. Jin, and F. Li. “The application of deep learning in computer vision.” In: *2017 Chinese Automation Congress (CAC)*. IEEE. 2017, pp. 6522–6527.
- [3] Z. Zou, Z. Shi, Y. Guo, and J. Ye. “Object Detection in 20 Years: A Survey.” In: *CoRR abs/1905.05055* (2019). arXiv: 1905.05055. URL: <http://arxiv.org/abs/1905.05055>.
- [4] J. Gleason, A. V. Nefian, X. Bouyssounousse, T. Fong, and G. Bebis. “Vehicle detection from aerial imagery.” In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 2065–2070. doi: 10.1109/ICRA.2011.5979853.
- [5] S. M. Azimi, R. Bahmanyar, C. Henry, and F. Kurz. “EAGLE: Large-Scale Vehicle Detection Dataset in Real-World Scenarios using Aerial Imagery.” In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 6920–6927. doi: 10.1109/ICPR48806.2021.9412353.
- [6] A. Ajay, V. Sowmya, and K. Soman. “Vehicle detection in aerial imagery using eigen features.” In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. 2017, pp. 1620–1624.
- [7] F. E. Nowruzi, P. Kapoor, D. Kolhatkar, F. A. Hassanat, R. Laganiere, and J. Rebut. “How much real data do we actually need: Analyzing object detection performance using synthetic and real data.” In: *CoRR abs/1907.07061* (2019). arXiv: 1907.07061. URL: <http://arxiv.org/abs/1907.07061>.
- [8] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. “Playing for Data: Ground Truth from Computer Games.” In: *Computer Vision – ECCV 2016*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Cham: Springer International Publishing, 2016, pp. 102–118. ISBN: 978-3-319-46475-6.
- [9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. “CARLA: An open urban driving simulator.” In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.
- [10] W. Qiu and A. Yuille. “Unrealcv: Connecting computer vision to unreal engine.” In: *European Conference on Computer Vision*. Springer. 2016, pp. 909–916.

- [11] S. Shah, D. Dey, C. Lovett, and A. Kapoor. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles." In: *CoRR* abs/1705.05065 (2017). arXiv: 1705.05065. URL: <http://arxiv.org/abs/1705.05065>.
- [12] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Bochoon, and S. Birchfield. "Training deep networks with synthetic data: Bridging the reality gap by domain randomization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 969–977.
- [13] C. Papageorgiou and T. Poggio. "A trainable system for object detection." In: *International journal of computer vision* 38.1 (2000), pp. 15–33.
- [14] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features." In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition*. CVPR 2001. Vol. 1. Ieee. 2001, pp. I–I.
- [15] M. Oualla, A. Sadiq, and S. Mbarki. "A survey of Haar-Like feature representation." In: *2014 International Conference on Multimedia Computing and Systems (ICMCS)*. 2014, pp. 1101–1106. DOI: 10.1109/ICMCS.2014.6911186.
- [16] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. "Pedestrian detection using wavelet templates." In: *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE. 1997, pp. 193–199.
- [17] J. Leitloff, S. Hinz, and U. Stilla. "Vehicle Detection in Very High Resolution Satellite Images of City Areas." In: *IEEE Transactions on Geoscience and Remote Sensing* 48.7 (2010), pp. 2795–2806. DOI: 10.1109/TGRS.2010.2043109.
- [18] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection." In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893.
- [19] S. Tuermer, F. Kurz, P. Reinartz, and U. Stilla. "Airborne vehicle detection in dense urban areas using HoG features and disparity maps." In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 6.6 (2013), pp. 2327–2337.
- [20] A. Kembhavi, D. Harwood, and L. S. Davis. "Vehicle detection using partial least squares." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.6 (2010), pp. 1250–1265.
- [21] P. Felzenszwalb, D. McAllester, and D. Ramanan. "A discriminatively trained, multiscale, deformable part model." In: *2008 IEEE conference on computer vision and pattern recognition*. Ieee. 2008, pp. 1–8.

- [22] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.
- [23] R. Girshick. "Fast R-CNN." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.
- [24] X. Chen, S. Xiang, C.-L. Liu, and C.-H. Pan. "Vehicle detection in satellite images by hybrid deep convolutional neural networks." In: *IEEE Geoscience and remote sensing letters* 11.10 (2014), pp. 1797–1801.
- [25] G. Cheng and J. Han. "A survey on object detection in optical remote sensing images." In: *ISPRS Journal of Photogrammetry and Remote sensing* 117 (2016), pp. 11–28.
- [26] J. Sivic and A. Zisserman. "Video Google: Efficient visual search of videos." In: *Toward category-level object recognition*. Springer, 2006, pp. 127–144.
- [27] J. Liu. "Image retrieval based on bag-of-words model." In: *arXiv preprint arXiv:1304.5168* (2013).
- [28] H. Zhou, L. Wei, C. Lim, D. Creighton, and S. Nahavandi. "Robust Vehicle Detection in Aerial Images Using Bag-of-Words and Orientation Aware Scanning." In: *IEEE Transactions on Geoscience and Remote Sensing* PP (July 2018), pp. 1–12. DOI: 10.1109/TGRS.2018.2848243.
- [29] Y.-G. Jiang, C.-W. Ngo, and J. Yang. "Towards optimal bag-of-features for object categorization and semantic video retrieval." In: *Proceedings of the 6th ACM international conference on Image and video retrieval*. 2007, pp. 494–501.
- [30] C. Cortes and V. Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297.
- [31] G. Mountrakis, J. Im, and C. Ogole. "Support vector machines in remote sensing: A review." In: *ISPRS Journal of Photogrammetry and Remote Sensing* 66.3 (2011), pp. 247–259.
- [32] Y. Freund. "Boosting a weak learning algorithm by majority." In: *Information and computation* 121.2 (1995), pp. 256–285.
- [33] Y. Freund, R. E. Schapire, et al. "Experiments with a new boosting algorithm." In: *icml*. Vol. 96. Citeseer. 1996, pp. 148–156.
- [34] T. Georgiou, Y. Liu, W. Chen, and M. Lew. "A survey of traditional and deep learning-based feature descriptors for high dimensional data in computer vision." In: *International Journal of Multimedia Information Retrieval* 9.3 (2020), pp. 135–170.



- [35] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. "Deep learning for visual understanding: A review." In: *Neurocomputing* 187 (2016), pp. 27–48.
- [36] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni. "Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3." In: *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*. IEEE. 2019, pp. 1–6.
- [37] K. He, X. Zhang, S. Ren, and J. Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition." In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.
- [38] S. Ren, K. He, R. Girshick, and J. Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems* 28 (2015).
- [39] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. "Ssd: Single shot multibox detector." In: *European conference on computer vision*. Springer. 2016, pp. 21–37.
- [40] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.
- [41] J. Redmon and A. Farhadi. "YOLO9000: better, faster, stronger." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.
- [42] J. Redmon and A. Farhadi. "Yolov3: An incremental improvement." In: *arXiv preprint arXiv:1804.02767* (2018).
- [43] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. "Microsoft coco: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [44] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. "DOTA: A large-scale dataset for object detection in aerial images." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3974–3983.
- [45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

- [46] S. Razakarivony and F. Jurie. "Vehicle detection in aerial imagery : A small target detection benchmark." In: *Journal of Visual Communication and Image Representation* 34 (2016), pp. 187–203. ISSN: 1047-3203. DOI: <https://doi.org/10.1016/j.jvcir.2015.11.002>. URL: <https://www.sciencedirect.com/science/article/pii/S1047320315002187>.
- [47] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng, and P. Yu. "Generalizing to unseen domains: A survey on domain generalization." In: *IEEE Transactions on Knowledge and Data Engineering* (2022).
- [48] H. Abu Alhajja, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. "Augmented reality meets computer vision: Efficient data generation for urban driving scenes." In: *International Journal of Computer Vision* 126.9 (2018), pp. 961–972.
- [49] W. Liu, J. Liu, and B. Luo. "Can Synthetic Data Improve Object Detection Results for Remote Sensing Images?" In: *CoRR abs/2006.05015* (2020). arXiv: 2006.05015. URL: <https://arxiv.org/abs/2006.05015>.
- [50] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial networks." In: *Communications of the ACM* 63.11 (2020), pp. 139–144.
- [51] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification." In: *Neurocomputing* 321 (2018), pp. 321–331.
- [52] L. Abady, M. Barni, A. Garzelli, and B. Tondi. "GAN generation of synthetic multispectral satellite images." In: *Image and Signal Processing for Remote Sensing XXVI*. Vol. 11533. SPIE. 2020, pp. 122–133.
- [53] S. Sharafi, B. Majidi, and A. Movaghar. "Low altitude aerial scene synthesis using generative adversarial networks for autonomous natural resource management." In: *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*. IEEE. 2019, pp. 322–326.
- [54] A. Tsirikoglou, J. Kronander, M. Wrenninge, and J. Unger. "Procedural modeling and physically based rendering for synthetic data generation in automotive applications." In: *arXiv preprint arXiv:1710.06270* (2017).
- [55] M. Wrenninge and J. Unger. "Synscapes: A photorealistic synthetic dataset for street scene parsing." In: *arXiv preprint arXiv:1810.08705* (2018).
- [56] S. R. Richter, Z. Hayder, and V. Koltun. "Playing for benchmarks." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2213–2222.

- [57] L. Laux, S. Schirmer, S. Schopferer, and J. Dauer. "Build Your Own Training Data - Synthetic Data for Object Detection in Aerial Images." In: *Software Engineering 2022 Workshops*. Ed. by J. Michael, J. Pfeiffer, and A. Wortmann. Bonn: Gesellschaft für Informatik e.V., 2022, pp. 182–190. DOI: 10.18420/se2022-ws-18.
- [58] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield. "Structured domain randomization: Bridging the reality gap by context-aware synthetic data." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7249–7255.
- [59] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan. "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" In: *arXiv preprint arXiv:1610.01983* (2016).
- [60] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3234–3243.
- [61] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. "Virtual worlds as proxy for multi-object tracking analysis." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4340–4349.
- [62] F. Kong, B. Huang, K. Bradbury, and J. Malof. "The Synthinel-1 dataset: A collection of high resolution synthetic overhead imagery for building segmentation." In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 1814–1823.
- [63] L. Chen, F. Liu, Y. Zhao, W. Wang, X. Yuan, and J. Zhu. "Valid: A comprehensive virtual aerial image dataset." In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020, pp. 2009–2016.
- [64] L. Jing and Y. Tian. "Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (2021), pp. 4037–4058. DOI: 10.1109/TPAMI.2020.2992393.
- [65] N. Koenig and A. Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.
- [66] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, et al. "Unity: A general platform for intelligent agents." In: *arXiv preprint arXiv:1809.02627* (2018).

- [67] *Unreal Engine*. Retrieved October 10, 2022. URL: <https://www.unrealengine.com/en-US/unreal-engine-5>.
- [68] *LoD3 Road Space Models*. Retrieved October 14, 2022. URL: <https://github.com/savenow/lod3-road-space-models>.
- [69] *CityGML*. Retrieved October 15, 2022. URL: <https://www.ogc.org/standards/citygml>.
- [70] *ASAM OpenDRIVE*. Retrieved October 15, 2022. URL: <https://www.asam.net/standards/detail/opendrive/>.
- [71] S. Wu, L. Liebel, and M. Körner. "Derivation of Geometrically and Semantically Annotated UAV Datasets at Large Scales from 3D City Models." In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 4712–4719. DOI: 10.1109/ICPR48806.2021.9412256.
- [72] *City Park Environment Collection*. Retrieved October 10, 2022. URL: <https://www.unrealengine.com/marketplace/en-US/product/city-park-environment-collection>.
- [73] *Unreal Engine Marketplace*. Retrieved October 14, 2022. URL: <https://unrealengine.com/marketplace/en-US/store>.
- [74] C. Quiter and M. Ernst. *deepdrive/deepdrive: 2.0*. Retrieved October 10, 2022. URL: <https://doi.org/10.5281/zenodo.1248998>.
- [75] *Unreal Python API Documentation*. Retrieved October 17, 2022. URL: <https://docs.unrealengine.com/5.0/en-US/PythonAPI/>.
- [76] *Python Script Plugin*. Retrieved October 17, 2022. URL: <https://docs.unrealengine.com/5.0/en-US/API/Plugins/PythonScriptPlugin/>.
- [77] *Vehicle Variety Pack*. Retrieved October 17, 2022. URL: <https://www.unrealengine.com/marketplace/en-US/product/bbcb90a03f844edbb20c8b89ee16ea32>.
- [78] *Vehicle Variety Pack Volume 2*. Retrieved October 18, 2022. URL: <https://www.unrealengine.com/marketplace/en-US/product/9a705589d1994c6e8757fdbedaf698af>.
- [79] *Blueprint Overview*. Retrieved October 17, 2022. URL: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/>.
- [80] *Collision Overview*. Retrieved October 17, 2022. URL: <https://docs.unrealengine.com/5.0/en-US/collision-in-unreal-engine---overview/>.
- [81] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo. "Object detection using convolutional neural networks." In: *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE. 2018, pp. 2023–2027.

- [82] J. Han, J. Ding, N. Xue, and G.-S. Xia. “Redet: A rotation-equivariant detector for aerial object detection.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2786–2795.
- [83] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. “Focal loss for dense object detection.” In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.
- [84] P. Dollár, C. Wojek, B. Schiele, and P. Perona. “Pedestrian detection: A benchmark.” In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 304–311.
- [85] *True Positive Rate*. Retrieved October 14, 2022. URL: <https://deepchecks.com/glossary/true-positive-rate/>.
- [86] *False Positive Rate*. Retrieved October 14, 2022. URL: <https://deepchecks.com/glossary/false-positive-rate/>.
- [87] *Precision-Recall*. Retrieved October 14, 2022. URL: [https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_precision\\_recall.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html).
- [88] *PyImageSearch*. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. Retrieved October 10, 2022.
- [89] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. “The pascal visual object classes (voc) challenge.” In: *International journal of computer vision* 88.2 (2010), pp. 303–338.