Enhancing Drone Surveillance with NeRF: Real-World Applications and Simulated Environments

Joakim Lindén*† Giovanni Burresi* Håkan Forsberg* Masoud Daneshtalab*
Ingemar Söderquist†‡

* *School of Innovation, Design and Engineering, Division of Intelligent Future Technologies, Mälardalen University, Västerås, Sweden*
† *Saab Aeronautics, Stockholm, Sweden*
‡ *School of Electrical Engineering and Computer Science, Division of Electronics and Embedded Systems, Royal Institute of Technology, Stockholm, Sweden*

***Abstract*—Machine Learning (ML) systems require representative and diverse datasets to accurately learn the objective task. In supervised learning data needs to be accurately annotated, which is an expensive and error-prone process. We present a method for generating synthetic data tailored to the use-case achieving excellent performance in a real-world usecase. We provide a method for producing automatically annotated synthetic visual data of multirotor unmanned aerial vehicles (UAV) and other airborne objects in a simulated environment with a high degree of scene diversity, from collection of 3D models to generation of annotated synthetic datasets (synthsets). In our data generation framework SynRender we introduce a novel method of using Neural Radiance Field (NeRF) methods to capture photo-realistic high-fidelity 3D-models of multirotor UAVs in order to automate data generation for an object detection task in diverse environments. By producing data tailored to the real-world setting, our NeRF-derived results show an advantage over generic 3D asset collection-based methods where the domain gap between the simulated and real-world is unacceptably large. In the spirit of keeping research open and accessible to the research community we release our dataset VISER DroneDiversity used in this project, where visual images, annotated boxes, instance segmentation and depth maps are all generated for each image sample.**

***Index Terms*—datasets, neural networks, synthetic data generation, automatic annotation, dataset generation**

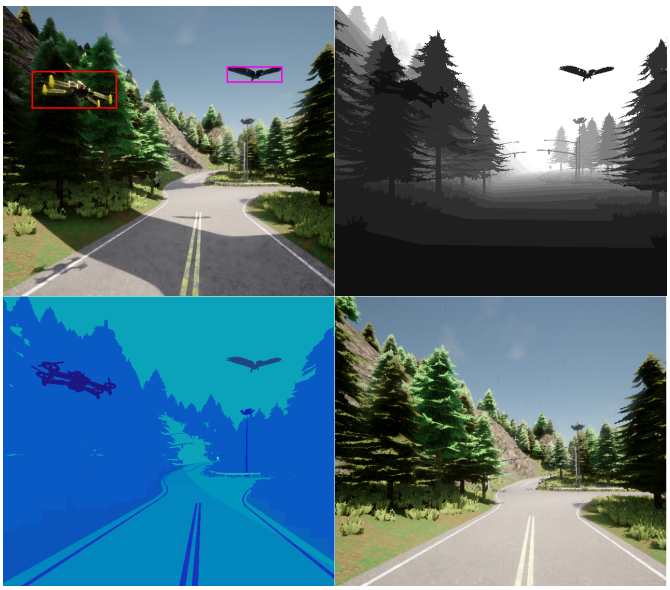

Fig. 1. Sample of images from dataset Viser DroneDiversity. Top left: RGB image with two annotated objects. Top right: Depth map (exaggerated for illustration purposes). Bottom left: Instance segmentation image. Bottom right: Background image where objects have been removed. Best viewed in colour.

I. Introduction

There are considerable changes going on in aviation airspace management, where U-space is a framework developed by the European Union Aviation Safety Agency (EASA) and supported by the Single European Sky ATM Research (SESAR) Joint Undertaking, designed to facilitate integration of UAVs safely into the airspace. It encompasses a set of services and procedures aimed at managing UAV traffic in a given airspace, especially at low altitudes. This framework is essential for enabling complex drone operations beyond visual line of sight (BVLOS) by providing a standardized system for airspace access, UAV localization, data exchange, and collision avoidance. Globally, the International Civil Aviation Organization (ICAO), a specialized agency of the United Nations, also plays a significant role in setting international standards and recommended practices for civil aviation, including integrating unmanned aerial system (UAS) into the global airspace. Surveillance systems for this cause, specifically real-time tracking of UAVs in confined airspace, are important parts of this concept. Here, our project Visual Inspection of airspace for air traffic safety and SEcuRity (VISER) aims at providing a scalable solution for a networked system of visual cameras for monitoring a particular area of interest (e.g. critical infrastructure) where the objectives of the system are to determine the position in 3D space of potential

We acknowledge the support of the Swedish Knowledge Foundation via the RELIANT research school, grant no 20220130, via Sweden's Innovation Agency and the Swedish Foundation for Strategic Research.

threats and classify them in type of UAV or dismiss them as non-invasive naturally occurring objects (e.g. birds). These findings will be handed to a U-Space Service Provider (USSP) for further processing and decision-making.

In this paper, we tackle the problem of visual surveillance of critical infrastructure by detecting UAVs, e.g. multirotor drones in a specific airspace. The object detection problem is not new, however, we are addressing the problem using only minimal amounts of annotated real captured data, and these are used only for test purposes. The way we approach this problem is by introducing a digital simulation world in which we place the objects we wish to detect. Programmatic control over a wide range of environment aspects and parameters enables controlled data generation for our object detection task. Automating the data generation facilitates the growth of the annotated synthetic datasets (synthsets) over time and allows for a quick turn-around time for exploring different simulation configurations and their impact on the detection task. We also generate all the annotation meta data, assuring it is always in sync with the visual representations of the scene. Another important reason for doing this is to be able to scale the solution, allowing for future growth by mining extensions to synthsets in an iterative way.

This paper is organized as follows: In Section II we present relevant related work. In Section III we explain our method for creating the synthsets from sourcing 3D assets to finalized synthset. In Section IV we describe the main properties of our datasets, including annotation meta-data, illustrating how they can be built upon in future work. In Section V we utilise our generated synthsets to train detection models in different configurations. In section VI we present the results from our experiments, including a statistical analysis of accuracy improvements. In Section VII we discuss the results and how to interpret the findings. Finally, Section VIII concludes this paper and layout possible future extensions of this work.

II. Related work

Synthetic data can be generated in multiple ways for different purposes [1]–[5]. Suter and Nuesch [6] and Zdziebko and Holak [7] have explored the use of simulators in generating synthetic visual data. Suter and Nuesch focused on the automated generation of landscape visualization databases using remote sensing and Geographic Information Systems (GIS), while Zdziebko and Holak used the finite element method and Blender graphics program [8] to render synthetic images of mechanical structures. Wang et al. [9] presented a scheme for generating and adapting synthetic images for object detection in smart vending machines, where they considered the simulation of cluttered objects and wide-angle cameras. Their environment was well-defined and they focused on the foreground objects under different lighting conditions. They also included a style-transfer adaptation step primarily to the foreground objects to increase photo-realism and a geometry deformation procedure to increase diversity in the objects' appearance.

Adams et al. [10] provided a review of the qualities of synthetic visual data production, where they highlighted the use of filtering, augmentation, and object domain randomization techniques to improve accuracy, and concluded that the problems related to labeling, image quality and privacy issues where clearly reduced as the bottleneck of manual data annotation was removed.

Akyon et al. [11] took on the Drone-vs-Bird Challenge and their solution was largely based on two things - they added synthetic data to expand the provided annotated dataset and they applied a tracking Kalman filter in the temporal domain. Their synthetic data were, however, limited to inserting models of drones onto a 2D image background. Although that might have been sufficient for distant drone detection, it will not handle occlusions due to terrain in a good way. They also implied that the domain gap between synthetic and real sets was so large that only small amounts of synthetic data could be used. We use Drone-vs-Bird dataset for parts of the evaluation.

Vin et al. [12] recently released an updated version of the Scenic probabilistic programming language, which allows for describing scenarios with objects scattered in 3D space. Scenic enables the user to programmatically define a scenario where object pose and position, as well as camera view-point, are randomized in a controlled way. Furthermore, it provides scene randomization and accurate occlusion detection using ray-tracing techniques. We leverage Scenic in our scene generation process since it ties well into the simulator of our choice.

Josifovski et al. [13] addressed the challenge of instance-based object detection and fine pose estimation in computer vision. Their proposed approach combined CNN robustness with fine-resolution 3D pose estimation using fully annotated synthetic data generated from 3D models (Blender). Results showed effective application on real-world images, and offered insights into training neural models with synthetic data for practical tasks like object grasping.

A study by Ros et al. [14] focused on vision-based semantic segmentation for autonomous driving in urban environments. The paper proposed using a virtual world to generate synthetic images with pixel-level annotations automatically. Their synthetic dataset, Synthia, was combined with real-world annotated images. The study demonstrated that incorporating Synthia in the training stage significantly enhanced the performance of deep convolutional neural networks (DCNNs) for semantic segmentation tasks.

In [15], Tremblay et al. employed domain randomization, where simulator parameters were intentionally randomized to enhance the network's ability to learn essential features. The study demonstrated that compelling performance could be achieved with non-artistically-generated synthetic data, and further fine-tuning on real data improved results beyond using real data alone. That approach offered a cost-effective solution for training neural networks, and reduced the reliance on large amounts of hand-annotated real-world data or high-fidelity synthetic worlds, addressing common bottlenecks in various applications. The evaluation focused on bounding box detection of cars using the KITTI [16] dataset. They

noted that the partial freezing of weights during fine-tuning decreased their performance instead of increasing it. This way (not freezing) is how we perform our fine-tuning. They also introduced distractors (i.e. flying random geometric shapes, not of interest to the detection function) in the environment.

In [17], Rajpura et al. implemented a transfer learning approach to address the challenge of detecting packaged food products in refrigerator scenes. It is interesting to note that achieving a high degree of photorealism in the synthetic images is not deemed crucial for achieving high performance levels (24% mAP across 55 categories). Furthermore, the authors observed a 12% boost by infusing a small set (10%) real images into the training dataset alongside the synthetic images. These findings highlighted the efficacy of the transfer learning strategy and showcased its potential for object detection tasks, even with limited annotated real-world data.

An open-source package [18] enhanced the Unity Editor and engine components, and facilitated the generation of perfectly annotated examples for various computer vision applications. Notably, the model trained with predominantly synthetic data surpassed the performance of a model trained solely on real data.

A very thorough review on the topic of synthetic data creation for computer vision was given by Paulin et al. [19], where they discussed the evolution and importance of synthsets for computer vision applications, focusing on representing dynamic environments. It provided an overview of synthsets, identifying and analyzing nine distinct methods and outlined a synthset generation diagram with 17 individual processes for creators to follow based on task requirements. They concluded geometric diversity within the objects of interest to be one of the more important parameters for a model to be able to generalize well to a given detection or classification problem. Furthermore they advised against using methods of generative adversarial training to create entire synthetic scenes because of problematic annotations. In our work we acknowledge this by focusing on scene diversity, object diversity and object photorealism using the Unreal Engine-based Carla Simulator [20].

On the topic of NeRF, Moon et al. [21] explored the use of NeRFs for data augmentation in defense applications, where they generated synthetic images for neural network learning. Unlike traditional image-based defense tech facing security issues, NeRF creates photo-realistic synthetic images without data screening. The study compared object detection performance with NeRF-generated images and live image learning, addressing potential performance issues. The paper highlighted NeRFs' effectiveness in augmenting data for defense-related neural network tasks.

The OMMO dataset [22] was introduced for evaluating NeRFs in tasks like novel view synthesis and surface reconstruction.

Li et al. [23] introduced Adv3D, a novel approach to modeling adversarial examples as NeRFs for 3D object detection in autonomous driving systems. Unlike traditional 2D pixel-level attacks, Adv3D leveraged NeRFs to generate more realistic and physically plausible adversarial examples in 3D space. The approach is trained by minimizing confidence in surrounding objects predicted by 3D detectors. The evaluation on an unseen validation set demonstrated significant performance reduction across different poses, scenes, and 3D detectors. The paper also proposed a defense method involving adversarial training through data augmentation.

Sattler et al. [24] developed a theoretical model for camera pose regression to understand the limitations of Convolutional Neural Network (CNN)-based camera pose regression problems, which in turn impacted the generalizability of rendering frameworks to unseen camera poses. Moreau et al. [25] acknowledged this in using NeRFs to improve robot relocalization. By incorporating a synthetic dataset generated by NeRF and strategically selecting virtual camera locations, their approach achieved a 60 percent reduction in error on datasets like Cambridge Landmarks and 7-scenes. The method enhanced pose regression accuracy without modifying the architecture, showcasing NeRFs' potential in advancing robot re-localization.

Meta-Sim [26] addressed the synthetic-to-real domain gap in machine learning by adapting synthetic scenes to match real-world dataset diversity for specific tasks. Using probabilistic scene grammars and optimizing non-differentiable objectives, Meta-Sim excelled in controlled settings and improved content generation quality for self-driving datasets.

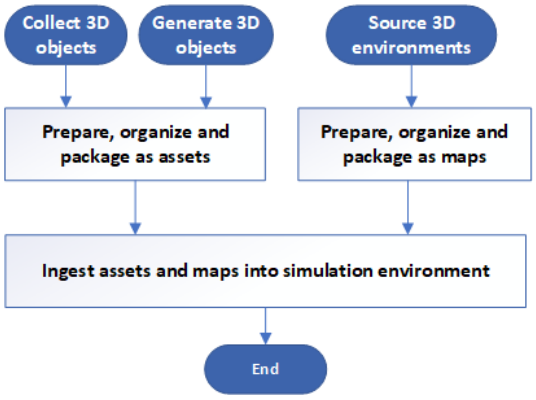

Fig. 2. Simulation assets preparation flow. We may start either by collecting existing 3D models or by generating our own such models using the NeRF generation flow. Map information can be sourced from OpenStreetMap, however several additional steps need to be taken to get a functioning simulation map.

III. Method: SynRender

For the given object detection task our automated data generation framework - SynRender - is composed of a number of sequential steps. Figure 2 outlines the principal steps of the process.

The main objective of implementing the SynRender method is to experiment with how our learned detection model is affected by the different ways we generate our datasets. Specifically we compare training on purely synthetic sets such as Synth Base described in section IV to training on additional synthetic datasets with NeRF-based objects infused. These trained detection models are validated mainly on realsets to

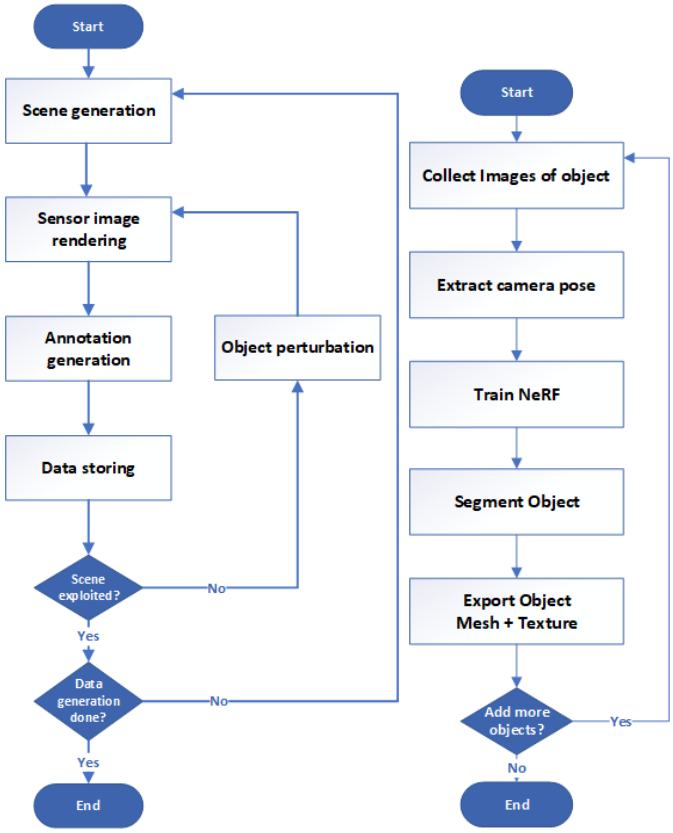

Fig. 3. Left: Data generation flow of SynRender. We iterate each scene $N = 10$ times before generating a new scene. We also removed all objects of interest and recorded the background image. Right: Flow for custom object generation in our simulator system setup. We use a tool to create a digital twin 3D-object from video recordings. Objects are segmented out from the background, and the model is exported as a mesh with draping textures. The models generated in this process are input to the simulation preparation flow shown in Figure 2.

capture the authenticity of real world use-cases and not remain completely in the simulated world.

We target in this work the specific simulator Carla [20] but the process is general and may be adopted to other simulators. The steps are outlined in the figures and referred to in the following subsections.

A. Assets preparation

We start out with the objects of interest, following either of two options:

a) Collect 3D objects: We can collect openly available 3D models of the objects. In our study, we want to detect different-sized UAVs, amongst other airborne objects such as aircraft and birds. We have used models that are openly available from online collections.

b) Generate 3D objects: Alternatively, we may choose to source our 3D models from another process, outlined in Figure 3 (right) where NeRF training learns to represent a full 3D object using photos as input. There are several methods of automating the creation of virtual 3D objects based on real-world examples, sometimes referred to as digital twins. Photogrammetry is a commonly used method based on collecting images, registering them relative to each other (i.e. finding the camera poses) and using triangulation to deduce a mesh-based object optionally draped with textures. However when increased photo-realism and ease of use is preferred over absolute geometric accuracy of the deduced model, NeRF techniques are sometimes preferred. As in photogrammetry, we need to start with camera pose extraction (unless such information can be provided directly from the camera), which relies on the assumption that adjacent images overlap quite a bit. This means it is possible to find feature point correspondences between pairs of images. This will in turn make it possible to describe a rotation and translation matrix for each camera pose using homogeneous coordinates, relative to some fixed origin:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \quad \begin{matrix} R = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi) \\ t \text{ translation vector.} \end{matrix} \tag{1}$$

This matrix transforms coordinates between the world frame and the camera frame of reference. The rotations are applied in the order yaw(ψ), pitch(θ) and finally roll (ϕ). The NeRF training process essentially maps 3D coordinates and viewing angles to colour and density distributions along rays of light. A fully connected deep neural network is trained on this mapping problem. The photometric loss function $\mathcal{L}$ is taken to be the difference in colours between rendered and observed pixel for each generated ray in direction $\mathbf{d}$ from the camera origin $\mathbf{o}$.

$$\mathbf{r}(\lambda) = \mathbf{o} + \lambda \mathbf{d} \tag{2}$$

$$\mathcal{L} = \sum_{r \in R} ||C(r) - C_{obs}(r)||^2 \tag{3}$$

Here R is the set of simulated rays, $C_{obs}(r)$ are the colours observed in the images. For more details on how the training of NeRFs are implemented, we refer to [27]. When training a NeRF to represent a real-world 3D object, we utilise LumaLabs' [28] service for generating high quality NeRFs using collections of images or video clips as input. For each NeRF-based model we collect a sequence of images of the object. From the images the camera pose is calculated and this information is used in the NeRF training process. From the trained NeRF model a segmentation is done separating the object from its background and the object is converted into a mesh with textures, all generated from the image data input. When all objects are independently generated in this way, each object is exported for use in our simulator (this flow is outlined in the right-hand part of Figure 3).

c) Source 3D Environments: In this step, we again have a choice to create our custom maps, using different methods like sourcing environment data for a particular area using, e.g., Open StreetMap [29], or re-use generic maps created by others for research purposes.

d) Prepare Assets: The 3D objects made available through the above inputs are imported into Blender [8]; objects are organized and named according to simulator guidelines and exported using the Filmbox (FBX) file format. Any manual

alteration or touch-ups may be applied here. A collection of object models is created in an object asset package. This is prepared in a format suitable for the simulation tool targeted. In our case, the asset package is prepared using the flow described in the Carla documentation [20]. Since we want to have segmentation label information included in our dataset generation, we need to build a custom distribution of the simulator with some minor amendments to the segmentation label definitions. With this customized simulator, we may properly prepare the asset packages from previous steps.

e) Prepare Maps: Custom map generation requires several additional steps to work, and hence, we limit our study to not include custom-generated maps. However it is the intention to extend in this direction in future studies.

f) Asset and map ingestion: Packages and maps are imported into the simulation environment and tested to assure all objects spawn correctly in the simulator.

B. Data generation

The task here is to generate data for our detection model training experiments. Figure 3 (left) outlines the steps included in the generation of synthsets.

1) Scenario generation: is the first step. We use Scenic [12] for automatically generating 3D scenes where objects of interest are placed procedurally in the scene according to a scenario description language. This allows for diverse scene rendering within specified parameters (e.g. min and max elevation of a multirotor UAV), but also camera placement, weather and daylight parameters. Scenic also checks the generated scene for consistency (e.g. objects do not occupy the same space) before it is finally rendered.

2) Sensor image rendering: is the next step, and here multiple virtual sensors are placed in the environment to capture several aspects of what is happening in the scene. Figure 1 shows one example of the sensor data recorded in our setup.

3) Annotation generation: is the next step. Here, we generate not only bounding boxes for the detection task but also save metadata on the global conditions of the scene (e.g. weather conditions). The bounding boxes are based on instance segmentation labels, allowing precisely fitting boxes.

4) Data Storing: is the final step in this iterative process, where we save all data. More information on the details of the dataset can be found in [30].

5) Object perturbation: is performed between successive image iterations, whereby we add a random shift to the pose and position of all our objects of interest. Our virtual camera pose is also slightly shifted for increased diversity. This means the background of the scene is largely unchanged, but the objects of interest are moving around.

When sufficient data has been collected, we stop the process. After this, we have a minor post-processing step, essentially converting from our internal annotation format to whatever is needed in the experiments.

IV. Datasets

In this section we provide details on the generated synthsets and otherwise sourced datasets (realsets).

A. Synthsets

In general, all synthsets are generated in the Carla simulator [20] and are structured in a consistent way. For each capture, we name all image and annotation files with a timestamp for ease of relating different sensor images. For each of the eight towns/maps included in Carla we sample 500 or 1000 images and additionally a number of background images (where objects of interest have been removed). The number of background images is equal to the number of unique scenes used for the set.

We include the following sensors and annotation:

- **RGB** camera, 24-bit RGB image, stored in Portable Network Graphics (PNG) format
- **Depth** image, 8-bit (each pixel value corresponds to the distance from the camera to the object to which the pixel belongs), stored in PNG format
- **Semantic instance segmentation** image, 24-bit (each pixel is encoded with a semantic label for the object type, but also a unique label for each instance of an object) with unique labels for our custom object types
- **Bounding box** annotation with unique labels for our custom object types, stored in JavaScript Object Notation (JSON) struct in a plaintext file
- **Scene metadata** including cloudiness, precipitation, precipitation deposits, wind intensity, solar position, fog density and distance and wetness, stored in JSON struct along with bounding box annotation in plaintext file
- **RGB + Bounding box** 24-bit RGB image, for convenience, we also record an RGB camera image with the bounding box annotation overlayed, stored in PNG format

In [30], we show more details concerning the dataset, including distribution of simulation parameters over the sets.

1) Synth Base: This synthset is generated with 4 categories of objects of interest: Commercial multirotor drones (6 different synthetic models), Fixed wing drones (2 larger military synthetic drone models), Commercial aeroplanes (1 synthetic model) and birds (7 synthetic models), see Figure 4. This is considered our baseline synthset. Each recorded snapshot in this dataset is rendered with a freshly generated scene, i.e. the scene is considered exploited after one generated snapshot and the object perturbation route in Figure 3 is never executed. This maximizes the scene diversity in the set, at the cost of generation time. Figure 6 shows an example of these images, of which we generate 500 per town.

2) NeRF Diverse: This synthset is generated with the same four categories of objects of interest, however the multirotor category objects are replaced with a total of four different NeRF-based models, see Figure 5, which carry a higher degree of photo realism compared to the synthetic multirotor models. Similarly to the Synth Base set, each recorded snapshot in this dataset is rendered with a freshly generated scene, see Figure 7. We generate 500 images per town.

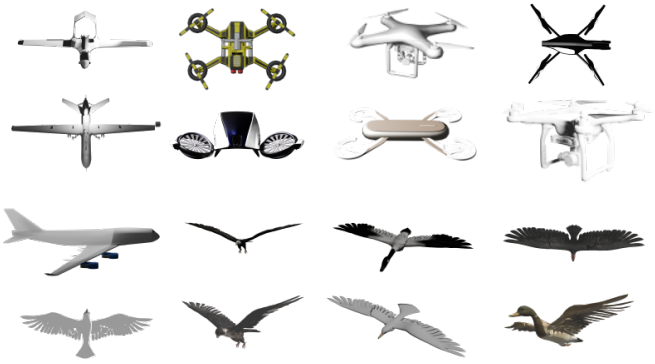

Fig. 4. Synthetic CAD-based assets. Leftmost column shows two fixed wing drones then an airliner-type aircraft. From the top-right corner we show the multirotor drones and the remaining thumbnails show birds. Note that objects are not to scale.

Fig. 5. Synthetic NeRF-based assets. Here we show the multirotor drones generated from NeRF models.

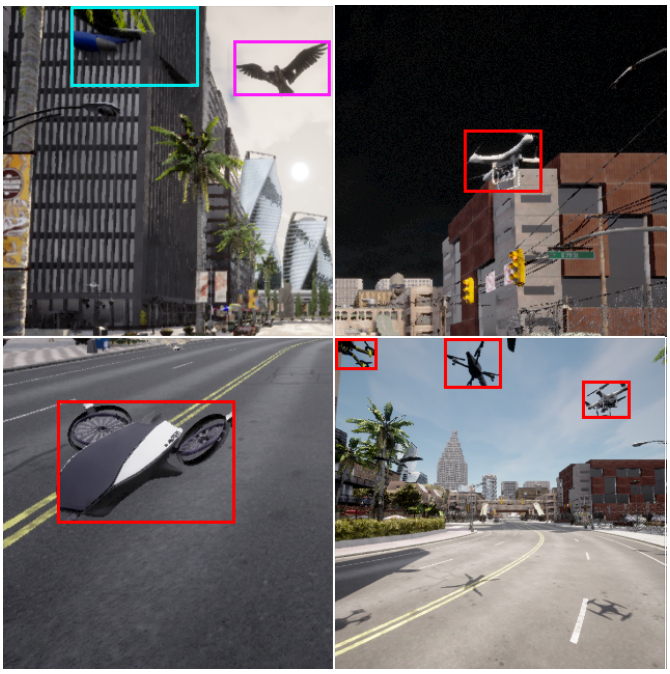

Fig. 6. Example of generated RGB image from Synth Base synthset. Note that the simulation environments generate shadows of our objects where applicable, however shadows remains un-annotated since they do not represent the real object. Best viewed in colour.

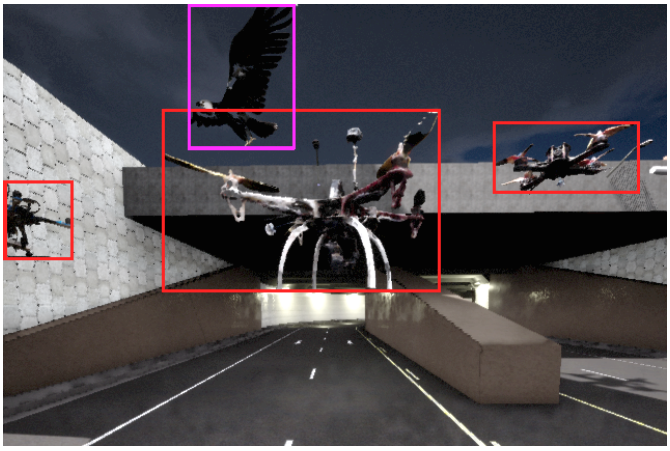

Fig. 7. Example of generated RGB image from NeRF Diverse synthset. Best viewed in colour.

3) NeRF Perturb: This synthset is generated with the same 4 categories of objects of interest (multirotors, birds, aircraft and fixed wings) with the same NeRF-based multirotor models. In this dataset, each scene is exploited 10 times (see Figure 3) before a new scene generation step is executed. The objects of interest are randomly moved around the scene, including object pose alteration. This yields a greatly reduced generation time. Because of the quicker generation time, we generated twice the amount of images in this set (i.e. 1000 images per town).

4) Swan Perturb: This synthset is generated with the same 4 categories of objects of interest, however in this set we target a specific multirotor model (the Swan), so only this NeRF-based model is in the multirotor object category. Similarly to NeRF Perturb, in this dataset each scene is exploited 10 times before a new scene generation step is executed. This dataset is a more targeted set to the detection of this specific multirotor model and is shown in Figure 8. Because of the quicker generation time, we generated twice the amount of images in this set (i.e. 1000 images per town).

B. Realsets

1) Drone-vs-Bird Subset: This is a real-world captured dataset, with video sequences of multirotor drones and birds originating from the WOSDETC 2023 Drone-vs-Bird detection challenge [31]. Only the drones are annotated. Since this dataset's original purpose was to target tiny drone detection, we sub-sample this dataset, leaving out the lower-resolution images. This subset serves as a real-world testing dataset in our experiments. However, we do also perform, in one experiment, some training on data from this source; therefore we have prepared two sets: DvB-train (randomly selecting 2456 samples from all images in the set) and DvB-test (for test) with 1235 images sampled as described above.

2) Field Swan: This is a real-world captured internal dataset, with one video sequence of a multirotor drone. The drone is a Swan-K1 drone from HEQUAV [32] and this recorded field realset is one of the test sets for our experiments.

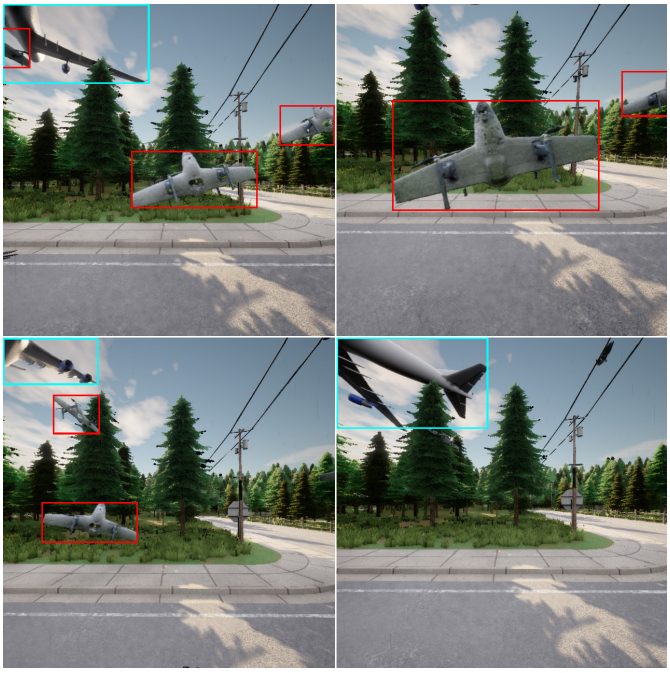

Fig. 8. Example of generated RGB image from Swan synthset. Best viewed in colour.

Fig. 9. Example of captured RGB image from field Swan realset. Best viewed in colour.

The annotation was done by us and includes 151 annotated images of a single category (multirotors). Figure 9 shows an example of this scene.

V. Experiments

In this section, we outline the experiments we performed to evaluate our data generation process. We also explain the real-world datasets (realsets) and synthetically produced synthsets used throughout the paper.

A. Experimental setup

In general, in our experiments, we apply a K-fold cross-validation setup, where our folds stem from the 8 different world maps we have in our simulator (i.e., $K = 8$). Specifically, we use 5 towns for training, two for validation, and one for testing and rotate this in K steps to get an estimate not only on model performance on a detection task but also the spread of ditto. For testing however, we also measure our performance results on two realsets (DvB-test and Field swan) for a more applicable target domain. We use Yolov8m as our base model provided by Ultralytics. It has been pre-trained on the MS-COCO [33] dataset.

B. Synthetic only

In this experiment we simply train on the Synth Base set. We train first for 100 epochs and in an extended experiment also on another 100 epochs. In these experiments we include only the artificially generated objects (i.e. no NeRF-based 3D-models).

C. NeRF Diverse fine-tuning

Here, we continue the model training from the result of experiment Synthetic only and fine-tune on the NeRF Diverse set. The tuning training is done for 100 epochs.

D. NeRF Perturb fine-tuning

In an attempt to increase image generation efficiency, we try to further our model training based on the NeRF Perturb set. the initial training is still based on the result of experiment Synthetic only and Similarly to the other experiments, the tuning training is done for 100 epochs. The idea behind this experiment is to see if emphasis on changing the objects' poses and positions yield sufficient diversity or if more scene diversity is beneficial.

E. Swan Perturb fine-tuning

For this study, we start our training from the result of experiment Synthetic only like in previous setups, however, the fine-tuning is done using the Swan Perturb set. Similarly to the other experiments, the tuning training is done for 100 epochs. This is done to see if we can successfully tailor a synthset towards a particular real-world scenario and achieve better performance this way.

F. Real world Drone-vs-Bird fine-tune

In this experiment, we start our training from the result of experiment Synthetic only and fine-tune on the Drone-vs-Bird Subset (dvb-train) set. Similarly to the other experiments, the tuning training is done for 100 epochs, however here we do this training on real-world data.

This experiment is included as a performance reference in the case that we have the full annotation of a real-world dataset. However, the purpose of this paper is not to produce the best possible model trained on labeled real data but rather to explore to what degree our automated data generation method keeps up with this method.

VI. Results

In this section, we show our experimental results. The results of the following sub-sections will be analyzed in following sections.

We measure how well an object detection model performs using a commonly used accuracy measure called mean Average Precision (mAP). This measure usually comes in two different flavours, mAP and mean Average Precision @ 50% IoU (mAP50). mAP50 requires an Intersection over Union (IoU) of at least 50% between the true and predicted bounding box (i.e. the boxes need to overlap by at least 50%) to count a detection as correctly classified. For mAP we consider a range of IoU from 50% up to 95%. The effect this will have on the metric is that mAP50 will be less sensitive (more forgiving) to errors in predicted box position.

In Figure 10 we first show how all our trained models perform relative to each other on synthetic test data. This shows that our training process is able to learn quite well the content of our data, but it does not show how well the models generalize. We show this for multirotor and bird classes here, but in the following diagrams we focus only on the multirotor class since it is the only class where we have annotation across all test sets. Every box in the boxplot diagrams with a distinct colour represents an experiment where a collection of models were trained on a particular type of data, given by the legend. In Figure 11 we show how our trained models perform on the realset Drone-vs-Birds test set DvB-test on the task of multirotor detection. In Figure 12 we see how our models perform on our own real world test scenario.

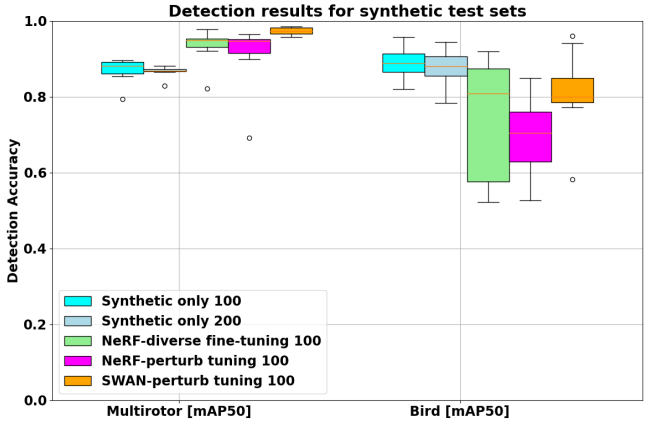

Fig. 10. Results from detection models trained of different synthsets, detailed by the legend. The models are tested on unseen test data from the same source as the training data (e.g. the 'SWAN-perturb tuning 100' experiment is tested on test data from the Field Swan synthset). Best viewed in colour.

A. Statistical tests

We perform a Wilcoxon signed rank test [34] to determine whether or not our achieved NeRF-model-based results outperform those where only synthetic content is present. Our test is setup with the hypothesis that the reference experiment (Synthetic only base experiment) mAP results are equally distributed to those of the various experiments we subject to this test. By rejecting the hypothesis with a p-value < 0.05 we may conclude that there is a statistically significant difference in the distribution of the reference and test experiment distributions,

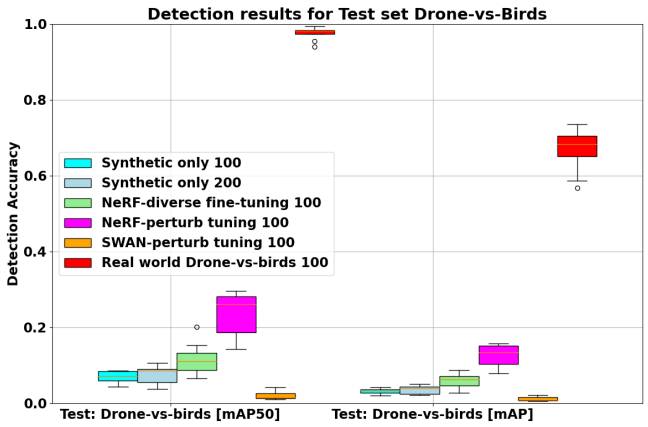

Fig. 11. Results from detection models trained of different synthsets, detailed by the legend (except the red boxes, representing a model fine-tuned on real data). The models are tested on unseen test data from the Drone-vs-Birds subset. Best viewed in colour.

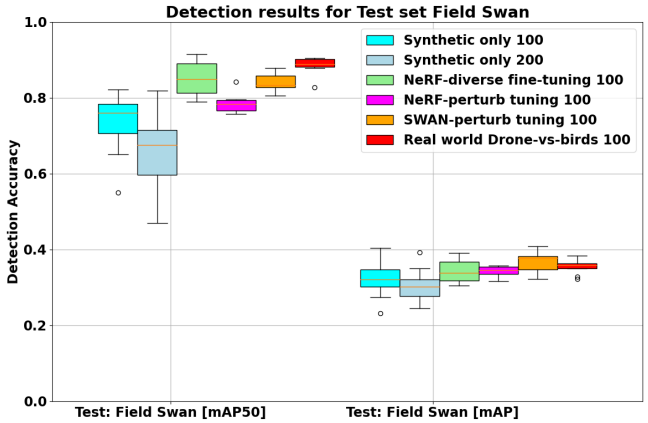

Fig. 12. Results from detection models trained of different synthsets, detailed by the legend. The models are tested on unseen test data from the Field Swan realset. Best viewed in colour.

with $> 95\%$ confidence level. In Table I and Table II, we see the median results from our cross-validation experiments for mAP50 and mAP metrics, respectively. The tables also show which experiments yielded results superior to those of the synthetic-only reference. These are marked in **bold face**.

VII. Discussion

Here we discuss the results of our experiments.

A. Synthset accuracy

Figure 10 shows that the detection accuracy is rather high for the multirotor class across all different experiments, with a notably higher level for the NeRF-based synthsets. In this case, we are validating on synthsets which are in-distribution with the training data for each case, meaning the domain gap is not an issue here. However for the bird class where we have no NeRF-based bird objects we notice a drop in performance. Likely this is due to the limited number of annotated objects in the bird class, compared to the multirotor class. This in

TABLE I
RESULTS FOR MULTIROTOR CLASS MAP50 ACCURACY. ROWS IN BOLD INDICATE EXCEEDING THE SYNTHETIC ONLY 100 BASE EXPERIMENT WITH $> 95\%$ CONFIDENCE LEVEL. P-VALUE COLUMN SHOWS THE PROBABILITY OF THE RESULT BEING EQUAL TO THE REFERENCE.

Test Split	Experiment name	median mAP50	P-value
Synthsets	Synthetic only 100	0.88	-
Synthsets	Synthetic only 200	0.87	0.74
Synthsets	**NeRF-diverse fine-tuning 100**	**0.95**	**0.01**
Synthsets	NeRF-perturb tuning 100	0.95	0.15
Synthsets	**SWAN-perturb tuning 100**	**0.97**	**0.01**
Synthsets	Real world Drone-vs-birds 100	0.07	0.01
Drone-vs-bird	Synthetic only 100	0.07	-
Drone-vs-bird	Synthetic only 200	0.09	0.46
Drone-vs-bird	**NeRF-diverse fine-tuning 100**	**0.11**	**0.02**
Drone-vs-bird	**NeRF-perturb tuning 100**	**0.26**	**0.01**
Drone-vs-bird	SWAN-perturb tuning 100	0.02	0.01
Drone-vs-bird	**Real world Drone-vs-birds 100**	**0.98**	**0.01**
Field Swan	Synthetic only 100	0.76	-
Field Swan	Synthetic only 200	0.68	0.15
Field Swan	**NeRF-diverse fine-tuning 100**	**0.85**	**0.01**
Field Swan	NeRF-perturb tuning 100	0.78	0.20
Field Swan	**SWAN-perturb tuning 100**	**0.83**	**0.01**
Field Swan	**Real world Drone-vs-birds 100**	**0.89**	**0.01**

TABLE II
RESULTS FOR MULTIROTOR CLASS MAP ACCURACY. ROWS IN BOLD INDICATE EXCEEDING THE SYNTHETIC ONLY 100 BASE EXPERIMENT WITH $> 95\%$ CONFIDENCE LEVEL. P-VALUE COLUMN SHOWS THE PROBABILITY OF THE RESULT BEING EQUAL TO THE REFERENCE.

Test Split	Experiment name	median mAP	P-value
Synthsets	Synthetic only 100	0.73	-
Synthsets	Synthetic only 200	0.74	0.15
Synthsets	**NeRF-diverse fine-tuning 100**	**0.80**	**0.05**
Synthsets	NeRF-perturb tuning 100	0.81	0.46
Synthsets	**SWAN-perturb tuning 100**	**0.88**	**0.01**
Synthsets	Real world Drone-vs-birds 100	0.03	0.01
Drone-vs-bird	Synthetic only 100	0.03	-
Drone-vs-bird	Synthetic only 200	0.04	0.38
Drone-vs-bird	**NeRF-diverse fine-tuning 100**	**0.06**	**0.02**
Drone-vs-bird	**NeRF-perturb tuning 100**	**0.13**	**0.01**
Drone-vs-bird	SWAN-perturb tuning 100	0.01	0.02
Drone-vs-bird	**Real world Drone-vs-birds 100**	**0.68**	**0.01**
Field Swan	Synthetic only 100	0.32	-
Field Swan	Synthetic only 200	0.30	0.55
Field Swan	NeRF-diverse fine-tuning 100	0.34	0.46
Field Swan	NeRF-perturb tuning 100	0.35	0.64
Field Swan	**SWAN-perturb tuning 100**	**0.38**	**0.05**
Field Swan	Real world Drone-vs-birds 100	0.36	0.20

turn is due to the fact that when deriving the annotation bounding boxes we apply a minimum displayed size threshold which will exclude small and highly occluded objects from being annotated. Since the birds are the smallest objects of our assets, they are affected the most. The primary task is, however, to detect the multirotors correctly, and the birds may be thought of as distractor objects in this context. To increase the accuracy of the bird class, we could for instance impose class-specific size thresholds for the annotation of objects, thereby allowing smaller birds to be annotated and hence reducing class imbalance. This will at the same time help reduce the number of un-annotated birds present in the dataset, the presence of which will otherwise likely pollute the set with respect to the specific class.

B. Realset Accuracy

In Figure 11 the results are shown for both mAP50 and mAP metrics across the experiments. We note a lower level of accuracy in general on realsets due to in part the domain gap between the synthsets and the Drone-vs-Bird test set (this gap is also apparent for the experiment where we fine-tune on real Drone-vs-Bird train set). However, when analysing the results of the experiments trained on our various synthsets we notice two things. Firstly, introducing NeRF-based multirotor objects in general boosts the performance (compared to synthetic only) and for the fine-tuning with NeRF-diverse and NeRF-perturb synthsets the boosts are statistically significant with 95% confidence levels. Secondly, we see that the result of the Swan-perturb fine-tuning experiment is significantly lower, which is expected since this synthset targets specifically the Swan multirotor drone which happens not to be included in the Drone-vs-Bird test set.

In Figure 12 we have a different situation. Here, the domain gap is less pronounced, and consequently, the accuracy on average is better for the detection task. Specifically for the mAP50 metric, we note that experiments including NeRF-based data seem to perform better, most of them significantly better (again with 95% confidence level) than the synthetic-only baseline. We note here that the Drone-vs-Bird experiment achieves performance just over our NeRF-based synthsets. In the reduced complexity Field Swan case we see that our method, despite not having to label a single image, still performs on a similar level to the model fine-tuned on annotated Drone-vs-Bird realset. Since the Swan-perturb synthset was created to be similar to its realset evaluation counterpart, it is expected to perform well in this setting.

VIII. CONCLUSIONS

We have proposed a method for generating synthetic datasets for use in machine learning objects detection and similar tasks. We outline the entire generation process, from the sourcing of models to generating the images for model training. We also include a process for using highly photo-realistic objects of interest using NeRF-techniques and ingesting such objects in a normal simulation asset management flow. We perform experiments on several variations of our synthetically produced datasets, the synthsets, validating the performance on two different real-world sets. We conclude that NeRF-based datasets perform significantly better on real-world data sets compared to the synthetic-only datasets. We also note that the domain gap between the realsets and synthsets is in the general case still large, which makes it challenging to use this process directly in a real-world scenario in its current state. Instead we foresee other methods of reducing the domain gap as being necessary to include in future extended versions of our presented workflow. The authors intend to pursue such an undertaking in future projects. Finally, we contribute our annotated synthsets to the community for different research

purposes to explore the possibilities of enhancing and streamlining ML model training endeavours.

1) Acknowledgements: The authors would like to thank members of the HERO group at Mälardalen University and colleagues at Saab for their valuable contributions in discussions.

REFERENCES

[1] B. Vanherle, S. Moonen, F. Van Reeth, and N. Michiels, "Analysis of training object detection models with synthetic data," *arXiv preprint arXiv:2211.16066*, 2022.

[2] E. Buls, R. Kadikis, R. Cacurs, and J. Ārents, "Generation of synthetic training data for object detection in piles," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041, pp. 533–540, SPIE, 2019.

[3] V. K. Berzin and M. Sudeykin, "Development of synthetic data generation algorithms for training neural network models for detecting objects in an image," *CPT2020 The 8th International Scientific Conference on Computing in Physics and Technology Proceedings*, 2020.

[4] J. Huh, K. Lee, I. Lee, and S. Lee, "A simple method on generating synthetic data for training real-time object detection networks," *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pp. 1518–1522, 2018.

[5] S. Hinterstoisser, O. Pauly, H. Heibel, M. Martina, and M. Bokeloh, "An annotation saved is an annotation earned: Using fully synthetic training for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pp. 0–0, 2019.

[6] M. Suter and D. Nuesch, "Automated generation of visual simulation databases using remote sensing and gis," in *Proceedings Visualization'95*, pp. 86–93, IEEE, 1995.

[7] P. Zdziebko and K. Holak, "Synthetic image generation using the finite element method and blender graphics program for modeling of vision-based measurement systems," *Sensors*, vol. 21, no. 18, p. 6046, 2021.

[8] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[9] K. Wang, F. Shi, W. Wang, Y. Nan, and S. Lian, "Synthetic data generation and adaption for object detection in smart vending machines," *arXiv preprint arXiv:1904.12294*, 2019.

[10] J. Adams, E. Murphy, J. Sutor, and A. Dodd, "Assessing the qualities of synthetic visual data production," in *2021 9th International Conference on Information and Education Technology (ICIET)*, pp. 452–455, IEEE, 2021.

[11] F. C. Akyon, O. Eryuksel, K. A. Ozfuttu, and S. O. Altinuc, "Track boosting and synthetic data aided drone detection," in *2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–5, IEEE, 2021.

[12] E. Vin, S. Kashiwa, M. Rhea, D. J. Fremont, E. Kim, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "3d environment modeling for falsification and beyond with scenic 3.0," in *International Conference on Computer Aided Verification*, pp. 253–265, Springer, 2023.

[13] J. Josifovski, M. Kerzel, C. Pregizer, L. Posniak, and S. Wermter, "Object detection and pose estimation based on convolutional neural networks trained with synthetic data," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 6269–6276, IEEE, 2018.

[14] 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*, pp. 3234–3243, 2016.

[15] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, 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*, pp. 969–977, 2018.

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[17] P. S. Rajpura, H. Bojinov, and R. S. Hegde, "Object detection using deep cnns trained on synthetic images," *arXiv preprint arXiv:1706.06782*, 2017.

[18] S. Borkman, A. Crespi, S. Dhakad, S. Ganguly, J. Hogins, Y.-C. Jhang, M. Kamalzadeh, B. Li, S. Leal, P. Parisi, *et al.*, "Unity perception: Generate synthetic data for computer vision," *arXiv preprint arXiv:2107.04259*, 2021.

[19] G. Paulin and M. Ivasic-Kos, "Review and analysis of synthetic dataset generation methods and techniques for application in computer vision," *Artificial Intelligence Review*, pp. 1–45, 2023.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

[21] S. Moon, J. Lee, J. Lee, D. Nam, and W. Yoo, "A study on nerf-based synthetic image generation and post-processing method for object detection," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1560–1562, IEEE, 2022.

[22] C. Lu, F. Yin, X. Chen, W. Liu, T. Chen, G. Yu, and J. Fan, "A large-scale outdoor multi-modal dataset and benchmark for novel view synthesis and implicit scene reconstruction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7557–7567, 2023.

[23] L. Li, Q. Lian, and Y.-C. Chen, "Adv3d: Generating 3d adversarial examples in driving scenarios with nerf," *arXiv preprint arXiv:2309.01351*, 2023.

[24] T. Sattler, Q. Zhou, M. Pollefeys, and L. Leal-Taixe, "Understanding the limitations of cnn-based absolute camera pose regression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3302–3312, 2019.

[25] A. Moreau, N. Piasco, D. Tsishkou, B. Stanciulescu, and A. de La Fortelle, "Lens: Localization enhanced by nerf synthesis," in *Conference on Robot Learning*, pp. 1347–1356, PMLR, 2022.

[26] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler, "Meta-sim: Learning to generate synthetic datasets," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4551–4560, 2019.

[27] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2021.

[28] Lumalabs AI, *Lumalabs AI*, 2023.

[29] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ." https://www.openstreetmap.org, 2017.

[30] "Viser drone diversity dataset." https://github.com/yoakim82/viser_drone_diversity, 2024.

[31] "6th international workshop on small-drone surveillance, detection and counteraction techniques (wosdetc) drone-vs-bird detection grand challenge." https://wosdetc2023.wordpress.com/, 2023.

[32] "Heq uav tech store." https://store.hequavtech.com/collections/heq-consumer-vtol, 2023.

[33] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[34] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics: Methodology and Distribution*, pp. 196–202, Springer, 1992.