



CHALMERS
UNIVERSITY OF TECHNOLOGY



Synthetic Data Generation for Industrial Perception Using 3DGS

A Simulation-Based Approach for Autonomous Mobile Robot Vision in Factory Environments

Master's thesis in Electrical Engineering

DINETH GURUSINGHE & YAHYA KHALID

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2026

www.chalmers.se

MASTER'S THESIS 2026

Synthetic Data Generation for Industrial Perception Using 3DGS

A Simulation-Based Approach for Autonomous Mobile Robot Vision
in Factory Environments

DINETH GURUSINGHE & YAHYA KHALID



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Synthetic Data Generation for Industrial Perception Using 3DGS
A Simulation-Based Approach for Autonomous Mobile Robot Vision in Factory
Environments
DINETH GURUSINGHE & YAHYA KHALID

© DINETH GURUSINGHE & YAHYA KHALID, 2026.

Supervisor: Erik Brorsson, Department of Electrical Engineering
Examiner: Knut Åkesson, Department of Electrical Engineering

Master's Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Screenshot of 3D Gaussian Splatting model used for this project.

Typeset in L^AT_EX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Synthetic Data Generation for Industrial Perception Using 3DGS
A Simulation-Based Approach for Autonomous Mobile Robot Vision in Factory
Environments

DINETH GURUSINGHE & YAHYA KHALID

Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous mobile robots operating in industrial environments require reliable camera-based perception to detect pedestrians, forklifts, and other vehicles in dynamic shared workspaces. However, collecting and manually annotating sufficiently large and diverse real-world datasets in factory environments is costly and time-consuming. This thesis investigates the use of 3D Gaussian Splatting (3DGS) as the basis of a synthetic-data-generation pipeline for training industrial object-detection models and examines factors affecting simulation-to-reality transfer.

A 3DGS reconstruction of a real factory environment was integrated into Unreal Engine as a digital-twin background and combined with dynamic simulated actors representing pedestrians, forklifts, and other vehicles. Synthetic images and corresponding object-detection annotations were generated from multiple virtual camera viewpoints and compiled into datasets for training YOLOv8-based detection models. A model trained on a 21-camera synthetic dataset achieved an mAP50 of 0.971 on synthetic test data but only 0.227 on real factory images, demonstrating a substantial simulation-to-reality domain gap.

Experiments showed that both camera-viewpoint diversity and environmental representation affected real-world generalisation. Models trained using the 3DGS factory representation transferred better to real images than those trained using a conventional Unreal Engine environment. However, because the appearance encoded by 3DGS is tied to the lighting conditions under which the scene was captured, lighting differences could not be varied independently, limiting the extent to which this improvement could be attributed solely to the background representation. Diffusion-based style transfer substantially improved zero-shot real-domain performance, increasing mAP50 from 0.125 to 0.440, while subsequent pseudo-labelling increased it further to 0.504 without requiring manually annotated real training images.

Synthetic pre-training followed by fine-tuning on real data increased mean mAP50 from 0.876 to 0.891 compared with real-only fine-tuning, although this difference fell within the observed variability across data splits. In contrast, joint training with style-adapted synthetic and real data produced more substantial gains when only limited annotated real data was available. Overall, the results indicate that 3DGS-based synthetic data is most valuable as a complement to real-world data rather than as a replacement for it, particularly when viewpoint diversity, visual domain alignment, and training strategy are considered together.

Keywords: synthetic data generation; 3D Gaussian Splatting; industrial perception; autonomous mobile robots; object detection; YOLO; simulation-to-reality; Unreal Engine; domain adaptation.

Acknowledgements

We would like to express our sincere gratitude to everyone who has supported us throughout the work on this master’s thesis.

First, we would like to thank our supervisor, Erik Brorsson, for the guidance, feedback, and encouragement provided during the project. The discussions and advice throughout the thesis work have been highly valuable in shaping the direction of the study and improving the quality of the work.

We would also like to thank our examiner, Knut Åkesson, for reviewing the thesis and providing academic guidance. We are grateful to Chalmers University of Technology for providing the opportunity to carry out this thesis and for the knowledge and skills gained during our studies.

We would also like to express our sincere gratitude to Kristofer Bengtsson for giving us the opportunity to undertake this master’s thesis and for selecting us for the project. We greatly appreciate the trust placed in us and are thankful for the chance to work on a project that has been both challenging and rewarding.

We would also like to thank Richard Andersen for their invaluable assistance in teaching us Unreal Engine and for providing helpful guidance and advice on a number of challenges encountered throughout the project.

Additionally, we would like to thank Björkman et al., authors of the bachelor’s thesis *Fotorealistisk simulering av mobila robotar i en fabriksmiljö* [1], which was carried out alongside our work. We are grateful to them for allowing us to use the 3D Gaussian Splatting model developed as part of their project.

Special thanks are also extended to Volvo Trucks for providing the industrial context and for enabling this work to be connected to a real factory environment. The opportunity to work on a practical and relevant problem in autonomous mobile robot perception and synthetic dataset generation provided valuable practical experience. Finally, we would like to thank our families and friends for their patience, support, and encouragement throughout our studies. Their support has been invaluable during the completion of this thesis.

AI Usage Statement

In the course of this study, artificial intelligence tools were used as supplementary aids for developing and refining scripts, as well as for improving the clarity, grammar, and structure of the written report. AI-generated suggestions were reviewed, modified, and verified by the authors before being incorporated into the work. All research procedures, experimental design, implementation, analysis, and conclusions were carried out and evaluated by the authors. The use of AI did not replace the authors’ academic judgment or responsibility for the content of this thesis.

Nomenclature

Abbreviations and Acronyms

3DGS	3D Gaussian Splatting
AMR	Autonomous mobile robot
COCO	Common Objects in Context (object-detection dataset and evaluation convention)
HSV	Hue, saturation, value
IoU	Intersection over Union
mAP	Mean Average Precision
RGB	Red, green and blue
SD	Standard deviation
sim-to-real	Simulation-to-reality; describes transfer from simulated to real-world data and the associated domain gap
SLAM	Simultaneous Localization and Mapping
UDA	Unsupervised domain adaptation
UE5	Unreal Engine 5
YOLO	You Only Look Once

Mathematical Symbols

3D Gaussian Splatting (Section 2.3)

$G_i(\mathbf{p})$	Value of the i th Gaussian primitive at spatial point $\mathbf{p}$
$\mathbf{p}$	Three-dimensional point at which a Gaussian is evaluated
μ_i	Centre (mean position) of the i th Gaussian
Σ_i	Covariance matrix of the i th Gaussian
α_i	Opacity parameter of the i th Gaussian
$\sigma(\cdot)$	Activation function applied to the opacity parameter
$\mathbf{R}$	Rotation matrix defining the Gaussian orientation
$\mathbf{S}$	Scale matrix defining the Gaussian dimensions

Camera Calibration and Projection (Section 4.5)

$\mathbf{c}^{\text{UE}}$	Camera position in Unreal Engine coordinates
t_x, t_y, t_z	Components of the Unreal Engine camera-position vector
P, Y, R	Original camera pitch, yaw and roll angles, respectively (not the projection or rotation matrices)
$\theta_p, \theta_y, \theta_r$	Pitch, yaw and roll angles after the sign-convention conversion
$R_x(\theta_r), R_y(\theta_p), R_z(\theta_y)$	Rotation matrices about the x -, y - and z -axes
R_{cw}^{UE}	Unreal Engine camera-to-world rotation matrix
R_{wc}^{UE}	Unreal Engine world-to-camera rotation matrix
$\mathbf{t}^{\text{UE}}$	Camera translation vector in Unreal Engine coordinates
$\mathbf{C}$	Coordinate-conversion matrix from Unreal Engine to OpenCV
$[R \mid t]^{\text{UE}}$	Camera extrinsic matrix expressed in Unreal Engine coordinates
$[R \mid t]^{\text{CV}}$	Camera extrinsic matrix expressed in OpenCV coordinates
$\mathbf{K}$	Intrinsic camera matrix
$\mathbf{P}$	Camera projection matrix, $\mathbf{K}[R \mid t]$
$\mathbf{R}, \mathbf{t}$	Extrinsic rotation matrix and translation vector (when used in camera projection)
$\mathbf{X}, \mathbf{X}_c$	Three-dimensional point in world coordinates and its position in camera coordinates
$\mathbf{X}_w, \tilde{\mathbf{X}}$	Homogeneous world-coordinate point used for projection
$\mathbf{x}, \tilde{\mathbf{x}}$	Homogeneous projected image point
$\tilde{u}, \tilde{v}, \tilde{w}$ (or u', v', w')	Components of a projected point before perspective division
u, v	Pixel coordinates after perspective division
Z_c	Depth of a 3D point in the camera coordinate system

Bounding-Box Conversion and Model Training (Section 4.5–4.6)

u_i, v_i	Projected pixel coordinates of the i th 3D bounding-box corner
$x_{\min}, x_{\max}$	Leftmost and rightmost pixel coordinates of a 2D bounding box
$y_{\min}, y_{\max}$	Topmost and bottommost pixel coordinates of a 2D bounding box
w_b, h_b	Original 2D bounding-box width and height
w'_b, h'_b	Adjusted bounding-box width and height after artificial tightening
c	Object class ID (0: forklift; 1: other vehicle; 2: pedestrian)
x_c, y_c	Normalised bounding-box centre coordinates in the YOLO label format

w, h	Normalised bounding-box width and height in the YOLO label format
W, H	Image width and height in pixels
w_c	Classification-loss weight assigned to class c
n_c	Number of training instances belonging to class c

Visual Domain Adaptation and Result Comparisons (Sections 4.8 and 5)

x_s	Synthetic source image for image-to-image style transfer
x_t	Real target-domain reference image
p	Natural-language style-transfer instruction (as distinct from the 3D point $\mathbf{p}$ above)
G	Generative image model used for style transfer
$\hat{x}_{s \rightarrow t}$	Style-transferred image generated from synthetic source to real target appearance
Δ	Difference between compared performance results, such as the change in mAP50

Contents

Nomenclature	ix
Nomenclature	ix
List of Figures	xvii
List of Tables	xix
1 Introduction	1
1.1 Aim and Research Questions	2
1.2 Limitations	2
1.3 Main Contributions	3
1.4 Thesis Outline	3
1.5 Previous Work	4
1.6 Related Work	4
2 System Architecture and Main Theories	7
2.1 System Context and Purpose	7
2.1.1 Intended Use	7
2.2 System Overview	7
2.3 3D Gaussian Splatting	8
2.4 Data Augmentation	9
2.4.1 Color and Illumination Augmentations	9
2.4.1.1 Hue Adjustment	9
2.4.1.2 Saturation Adjustment	10
2.4.1.3 Brightness Adjustment	10
2.4.2 Geometric Transformations	11
2.4.2.1 Translation Augmentation	11
2.4.2.2 Scale Augmentation	11
2.4.2.3 Horizontal Flip Augmentation	12
2.4.2.4 Mosaic Augmentation	12
3 System Procedures	15
3.1 Data Gathering Procedure	15
3.1.1 General Approach	15
3.1.2 The Main Synthetic Image Generation Procedure	16
3.1.3 Moving Objects in the Simulation: Actors	17

3.1.4	Actor Movement Procedure	17
3.1.5	Actor Spawning Logic	18
3.2	Cut-and-Paste augmentations	18
3.2.1	Procedure with Simulated objects	18
3.2.2	Procedure with Real Objects	19
4	Implementation	23
4.1	Testing Dataset	23
4.1.1	Annotation Software	23
4.1.2	Testing Dataset Statistics	24
4.1.3	Unreal Engine	25
4.1.4	XGrids	25
4.2	Synthetic Actor Generation Using Generative Models	25
4.2.1	Motivation	26
4.2.2	Actor Appearance Generation with Sora	26
4.2.3	3D Asset Generation with Hunyuan3D	26
4.3	Simulation Construction and Main Procedure	27
4.3.1	Actor Model Animations	28
4.4	Resultant Data	29
4.5	Processing the Data	30
4.5.1	Camera Calibration Conversion	30
4.5.2	3D to 2D Bounding box Conversion	32
4.6	Computer Vision Model: YOLOv8	34
4.6.1	Model Setup and Hyperparameters	35
4.7	Cut-and-Paste Implementation	36
4.8	Visual Domain Adaptation using Generative Diffusion Models	37
5	Results	41
5.1	Evaluation Metrics	41
5.2	Dataset Overview	41
5.3	Baseline: Simulation-to-Real Domain Gap	42
5.3.1	Synthetic Model: In-Domain Performance	42
5.3.2	Synthetic Model: Performance on Real Test Set	43
5.4	Synthetic Dataset Design: Effect of Viewpoint Diversity	43
5.4.1	Single-Set Baselines	44
5.4.2	Cross-Set Evaluation	44
5.4.3	Combined Camera Sets	45
5.5	Synthetic Dataset Design: Background and Appearance	46
5.5.1	Gaussian vs. Non-Gaussian Background Rendering	46
5.5.2	Real Background Augmentation on Synthetic Dataset	48
5.5.3	Object Appearances vs. Background Representation	49
5.6	Bridging the Sim-to-Real Gap	51
5.6.1	Fine-tuning on Real Data: Baseline	51
5.6.2	Synthetic Pre-training Followed by Real Fine-tuning	52
5.6.3	Domain Adaptation via Style Transfer	53
5.6.4	Joint Training by Data Mixing	55
5.6.5	Pseudo-Labeling for Domain Adaptation	57

5.7	Summary of the Results	58
5.7.1	The Sim-to-Real Gap	58
5.7.2	Importance of Synthetic Dataset Design Factors	59
5.7.3	Domain Adaptation for Reducing Sim-to-Real Gap	60
5.7.4	Annotation Efficiency	61
6	Conclusion	63
7	Limitations, Ethics and Future Work	67
7.1	Ethics	68
7.2	Conclusion	69
	Bibliography	71
A	Appendix 1	I

List of Figures

2.1	Example of a 3D Gaussian Splatting reconstruction of an environment. Taken from [2]. The environment is represented using optimized 3D Gaussian primitives, with visible reconstruction artifacts in regions with limited or uncertain scene information. These artifacts are more pronounced for viewpoints for which the 3DGS model was not optimized.	8
2.2	Examples of hue augmentations taken from [3]	10
2.3	Examples of saturation augmentations taken from [3]	10
2.4	Examples of brightness augmentations taken from [3]	11
2.5	Examples of translation augmentations taken from [3]	11
2.6	Examples of scale augmentations taken from [3]	12
2.7	Example of Horizontal Flipping Augmentation. Taken from [3]	12
2.8	Example of Mosaic Augmentation. Useful for object detection and semantic segmentation. Taken from [3]	13
3.1	Diagram of the main approach used for this study. Generated with ChatGPT with some modifications	16
3.2	Synthetic image generation system overview	16
3.3	Cut-and-Paste procedure for simulated objects.	19
3.4	High-level overview of Cut-and-Paste dataset generation	20
4.1	Selection of the bounding-box location and size in a bird's-eye view. .	23
4.2	A preview of the bounding box can be seen in this window.	24
4.3	Instance distribution between classes	24
4.4	Summary of the overall statistics	25
4.5	Examples of appearances generated using Sora for human and vehicle actors in the factory.	27
4.6	Generative assets generation pipeline using Hunyuan3D and Sora . .	28
4.7	High-level overview of the simulation-environment construction process.	29
4.8	Synthetic images generated in the UE5 simulation from four different camera viewpoints for a random scene	29
4.9	An illustration of the Unreal Conventions. Taken directly from Unreal Engine application with added arrows for clarification.	30
4.10	Comparison of the original image (left) and the same image with 3D bounding-box annotations surrounding the detected actors (right). . .	32
4.11	Comparison of the original image (left) and the same image with 2D bounding-box annotations over the detected actors (right).	34

4.12	Illustration of the YOLOv8 architecture. Taken from [4]	35
4.13	Tool developed for image-to-image translation	38
4.14	Side-by-side comparison of a source image and a style-transferred image in which object appearances are adapted toward the real domain.	38
4.15	Side-by-side comparison of a source image and a style-transferred image with object geometry preserved.	39
5.1	Synthetic factory layout shown from a top-down view in the centre, surrounded by location specific RGB camera images from the factory.	42
5.2	A side-by-side comparison between 3DGS and corresponding mesh environments	47
5.3	A side-by-side comparison of real image and synthetic dataset augmented by it	48
5.4	Comparison of different background and object arrangements	49
5.5	A side-by-side comparison between source and style transferred image.	53
5.6	Visualizing sim-to-real domain gap.	58
5.7	Figure showing background and object appearance results side-by-side.	59
5.8	Progression of performance across different domain adaptation methods.	61
5.9	Effect of training strategy and availability of real annotated data on mAP50 evaluated on the real test set. Solid lines represent joint training with synthetic and real data, while dashed lines represent training using only real data or sequential synthetic pre-training followed by real fine-tuning.	62
A.1	Unreal Engine blueprint implementation for synthetic data collection.	I

List of Tables

5.1	Synthetic model performance on synthetic test set (in-domain baseline).	42
5.2	Synthetic model evaluated on the real-world test set (domain gap baseline).	43
5.3	Self-test baseline for each individual 7-camera synthetic set (train and test on same set).	44
5.4	Cross-evaluation matrix: mAP50 for all train/test set combinations. Diagonal entries are self-test baselines.	44
5.5	Per-class mAP50 for the weakest and strongest cross-set transfer conditions.	45
5.6	Cross-evaluation for combined 14-camera sets. Self-test results are also reported for completeness.	45
5.7	Evaluation between Gaussian and Non-Gaussian background datasets.	47
5.8	Evaluation of real augmentation vs. Gaussian Splatting	49
5.9	Models trained on four different combination of datasets evaluation metrics on real dataset	50
5.10	Real-only fine-tuning: mean $\pm$ SD across five random seeds (seeds 8, 34, 42, 77, 85). Test set size varies slightly per seed due to rounding of the 60/20/20 split.	51
5.11	Synth $\rightarrow$ Real fine-tuning: mean $\pm$ SD across five random seeds. Model initialised from 21-camera synthetic pre-trained weights before fine-tuning on real data.	52
5.12	Direct comparison of real-only fine-tuning vs. synth $\rightarrow$ real fine-tuning (mean mAP50 across five seeds). Δ = synth $\rightarrow$ real minus real-only.	52
5.13	Style transfer vs synthetic model evaluated on real test dataset.	53
5.14	Direct comparison of fine-tuning on synthetic dataset and equivalent style transferred dataset. Δ = mAP50 difference between synthetic vs. style transfer.	54
5.15	Performance of style transfer model on the real test set.	54
5.16	Performance of synthetic source model on the real test set.	54
5.17	Comparison of the style-transfer model and baseline YOLO models fine-tuned and evaluated on the real dataset.	55
5.18	Comparison of training strategies on the real test set. The reported splits refer to the real dataset only; the synthetic and style-transfer datasets were kept constant across experiments.	56

5.19 Pseudo-labelling results: style transfer model applied to unlabelled real images; high-confidence detections retained as pseudo-labels for further fine-tuning.	57
--	----

1

Introduction

According to Volvo [5], battery electric, internal-combustion, and hydrogen-powered trucks are assembled on a single production line at Volvo Trucks’ final assembly plants, which handle roughly 25,000 unique parts. It is stated in the article that, since it is infeasible to store all components directly at the line, a flexible internal logistics system is used to deliver materials just in time. To manage this, Volvo employs Autonomous Mobile Robots (AMRs) coordinated by a centralized, cloud-based control system. These robots operate in dynamic, human-shared environments and rely on ceiling-mounted cameras for perception. Using machine learning models for semantic segmentation, the system identifies obstacles to enable collision-free navigation. However, the article explains that the performance of such models depends on access to large, labeled datasets, which are expensive and time-consuming to collect in real-world conditions.

Wang and Deng [6] note that collecting and labeling real-world image data is often one of the most expensive and time-consuming stages of a machine learning project. They state that the process may require specialized equipment, access to operational factory environments, and expert annotators. Furthermore, collecting sufficiently varied examples of uncommon or safety-critical situations can be difficult. Methods that reduce the amount of real-world data required are therefore of considerable interest. According to Ultralytics documentation [3], data augmentation can increase the diversity of an existing dataset by applying transformations or noise to recorded samples, but it remains limited by the contents of the original dataset.

An alternative approach is to generate synthetic training data using a simulated environment. Once such an environment has been constructed, images and corresponding annotations can be generated automatically from multiple viewpoints and under controlled conditions. Synthetic data generation may therefore reduce the cost of data collection and enable greater control over scene composition, object placement, and environmental variation.

However, synthetic images often differ from images captured in real environments. Differences in geometry, lighting, textures, camera properties, and object appearance may cause models trained on synthetic data to perform poorly on real-world images. This discrepancy, detailed by Aoshima and Servin [7], is commonly referred to as the simulation-to-reality, or sim-to-real, gap. Identifying methods that reduce this gap remains an active area of research [8].

This thesis investigates the use of 3D Gaussian Splatting (3DGS) for generating synthetic image datasets in an industrial factory environment. 3DGS, described by Bernard et al. [9], represents and renders three-dimensional scenes using collections

of volumetric Gaussian primitives rather than conventional polygon-based geometry. In the proposed pipeline, a Gaussian-splat representation of a factory environment is imported into Unreal Engine and supplemented with dynamic actors, including pedestrians and industrial vehicles. Virtual cameras positioned throughout the environment are then used to collect images and annotations from multiple viewpoints. The generated dataset is used to train and evaluate a machine learning model for an industrial perception task. Based on the initial results, selected aspects of the simulation, dataset, and model configuration are modified to examine how they affect model performance.

1.1 Aim and Research Questions

The aim of this thesis is to investigate how 3D Gaussian Splatting can be used to generate data (synthetic images and labels) for training machine learning models to detect objects in RGB images in industrial factory environments. The work focuses on constructing a simulation pipeline in Unreal Engine, adding dynamic actors such as pedestrians and industrial vehicles, and evaluating how such synthetic data can support machine learning-based perception for autonomous mobile robots.

Based on this aim, the thesis addresses the following research questions:

1. Is it feasible to generate a synthetic dataset suitable for industrial object detection?
2. How effective are synthetic datasets for real-world industrial object detection?
3. How do dataset design choices, such as viewpoint diversity and background rendering style, affect object detection performance within the synthetic domain and across the simulation-to-reality domain?
4. To what extent does a simulation-to-reality domain gap exist, and how can it be reduced?

1.2 Limitations

This thesis focuses on 3DGS-based data generation in Unreal Engine and evaluates the generated dataset using one primary machine-learning model. The study uses Unreal Engine as its sole simulation platform. Although tests with other machine learning models were conducted, they did not produce meaningful results because of limitations of the 3D Gaussian Splatting model. No comparisons are made to other rendering technologies. The Unreal Engine implementation was developed using Blueprints, its visual programming interface. This approach was chosen to maintain consistency with previous work, which had been implemented using the same methodology. This resulted in longer dataset generation times [10]. Another limitation was that the work did not include constructing the 3D environment itself and instead relied on external assistance to obtain the 3D environment models. Finally, because of time and hardware constraints, some conclusions are supported by too few experiments for a fully quantitative assessment.

1.3 Main Contributions

Our main contributions are as follows:

1. Defined a pipeline that converts simulation output into a dataset usable by multi-view models or, with a simple script modification, by single-view models.
2. Identified key synthetic-dataset factors that affect the magnitude of the sim-to-real gap between the synthetic and real datasets.
3. Investigated multiple methods for bridging the sim-to-real gap and compared their effects on detection performance.

1.4 Thesis Outline

Chapter 2 presents the system architecture and the main theoretical concepts used throughout the thesis. The chapter describes the intended use and overall structure of the proposed system, introduces 3D Gaussian Splatting as the representation of the factory environment, and explains the data-augmentation techniques used during model training.

Chapter 3 describes the procedures developed for generating the synthetic dataset. This includes the image-capture process and solutions to rendering-related limitations associated with Gaussian Splatting in Unreal Engine. Additionally, the chapter describes the procedures used for actor movement and spawning logic, and the procedures used for Cut-and-Paste augmentation with both simulated and real objects. Chapter 4 details the implementation of the proposed pipeline. It describes the real-world test dataset and its annotation process, the Unreal Engine simulation environment, the generation and integration of synthetic actors, the construction of the simulation, and the resulting data format. The chapter also explains the conversion of Unreal Engine coordinates into camera and object-detection annotations, the YOLOv8 model configuration, the implementation of Cut-and-Paste augmentation, and the use of diffusion-based image-to-image translation for visual domain adaptation.

Chapter 5 presents and analyzes the experimental results. It first evaluates the performance of models trained and tested within the synthetic domain and then measures the synthetic-to-real domain gap using real factory images. Further experiments investigate the influence of camera-viewpoint diversity, background representation, object appearance, and real-background augmentation. The chapter also evaluates several approaches for reducing the domain gap, including real-data fine-tuning, synthetic pre-training followed by real fine-tuning, style transfer, joint synthetic–real training, and pseudo-labelling.

Chapter 6 summarizes the main findings of the thesis and relates the experimental results to the research questions, discussing the overall feasibility and value of using 3D Gaussian Splatting-based synthetic data for industrial object detection.

Chapter 7 discusses the limitations of the current approach and outlines directions for future work, including improvements to camera calibration, lighting control, domain randomization, object representation, style transfer, pseudo-labelling, and experimental evaluation.

1.5 Previous Work

This project builds on Ahmed et al. [11] who constructed the original 3D Gaussian Splatting reconstruction of the Volvo factory used to create the dataset. Specifically, they captured drone images of a factory hallway and used PostShot to construct a 3D Gaussian Splatting model. They also created a 3D Gaussian Splat model of the factory’s primary robot, which was used in this project. This project also builds on Albarham et al. [12], who extended Ahmed et al. [11] while shifting the focus toward evaluation of the Gaussian Splatting representations.

1.6 Related Work

Synthetic data generation has been widely studied as a means of reducing the cost and effort associated with collecting and manually annotating real-world image datasets. This is particularly relevant to computer vision tasks such as object detection and semantic segmentation, which often require large quantities of accurately labeled training data. Previous work has explored the use of virtual environments, computer games, and photorealistic scene reconstructions to generate synthetic images together with automatically produced annotations.

One central feature of synthetic datasets is automatic ground-truth generation. Ros et al. [13] introduced SYNTHIA, a large-scale dataset of synthetic urban images with pixel-level annotations for semantic segmentation. Richter et al. [14] similarly demonstrated that pixel-level semantic annotations could be extracted from realistic computer game environments. These studies show that virtual environments can provide large quantities of labeled data without requiring manual annotation. They also demonstrate that both built simulations and existing graphical environments can be used as sources of training data.

The realism and diversity of synthetic environments are also important considerations. SYNTHIA includes variations in urban scenes, weather conditions, seasons, and camera viewpoints, while the approach by Richter et al. uses the visual complexity of a commercial game environment. Such variations allow datasets to represent a wider range of scene configurations and viewing conditions. This is relevant for perception tasks in which models must handle changes in object placement, camera position, lighting, and occlusion.

More recent work has combined synthetic data generation with photorealistic scene-reconstruction methods. Deogan et al. [15] investigated the use of 3D Gaussian Splatting together with Unreal Engine for generating annotated datasets for autonomous mobile robots. Their approach integrates a 3D Gaussian Splatting reconstruction into a simulation environment and generates synthetic images for object detection. The study uses a virtual soccer field containing soccer balls and a robot model, demonstrating the use of 3DGS-based environments for robotics-oriented dataset generation.

The work is a continuation of the factory reconstruction studies by Ahmed et al. [11] and Albarham et al. [12]. While these earlier works provided and evaluated the Gaussian Splatting representation of the factory environment, this thesis extends

the reconstructed environment into a complete synthetic-data-generation and perception pipeline. Dynamic actors are integrated into the static 3DGS environment, including additional actor models created using generative methods, and the pipeline produces both 2D object-detection annotations and 3D object information from multiple synchronized viewpoints.

An important distinction from much of the related work is that the objective of this thesis is not limited to demonstrating that synthetic data can be generated. Instead, the generated data is evaluated against images from the corresponding real factory, allowing the simulation-to-reality gap to be measured directly. The experiments investigate factors that influence this gap, including camera-viewpoint diversity, background representation, and object appearance. In addition, several different methods for reducing the domain gap are evaluated, including synthetic pre-training followed by real-data fine-tuning, diffusion-based style transfer, joint training with synthetic and real data, and pseudo-labeling.

The contribution of this thesis is therefore positioned not as the introduction of a new rendering or synthetic-data-generation technique, but rather as an application- and evaluation-oriented extension of existing synthetic-data and 3DGS-based simulation research. Its main contribution is the integration of a 3DGS-based digital twin with dynamic industrial simulation, multi-camera data generation, automatic annotations, and systematic real-world evaluation in a practical factory perception setting. The results further position synthetic data not as a complete replacement for real-world training data, but as a complementary resource whose usefulness depends on careful dataset design and appropriate domain-adaptation strategies.

2

System Architecture and Main Theories

2.1 System Context and Purpose

The main objective of this system is to collect synthetic images from a working factory environment using multiple cameras placed throughout the environment at different angles. These synthetic images should ideally capture the environment with varying numbers of people and vehicles, resulting in multiple images where the background is the same, but the people and vehicles within the images differ.

2.1.1 Intended Use

The developed pipeline is intended to generate synthetic multi-camera RGB datasets for factory-automation perception tasks using a digital twin created with 3D Gaussian Splatting. However, this type of system can also be applied to other settings that use overhead cameras, such as retail and security-surveillance environments.

2.2 System Overview

For this project, a system was required that could simulate a physical environment in which different types of actors could be placed. These actors should all have behaviors that allow them to change position from scene to scene. Additionally, the environment should be viewable from multiple positions and angles which can be specified by the operator. Images that depict the environment from these viewpoints should be saved together with metadata identifying the viewpoint from which each image was captured. The positions of all actors and camera viewpoints should be saved for each scene. Additionally, the intrinsic and extrinsic camera parameters should be recorded for each viewpoint. The primary user input is the desired number of scenes to generate. For this project, the simulated environment is rendered using 3D Gaussian Splatting (3DGS). The use of 3DGS enables the environment to closely reproduce the appearance of the corresponding real-world scene. This high degree of visual similarity is particularly desirable for this work, as it enables the generated data to more accurately represent the visual characteristics of the real-world environment.

2.3 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) is a scene representation and rendering method used for photorealistic novel-view synthesis. A scene rendered using 3DGS is represented as a collection of three-dimensional Gaussian primitives. Each Gaussian stores properties such as position, scale, orientation, opacity, and color. The method was introduced by Kerbl et al. for real-time radiance-field rendering [9].

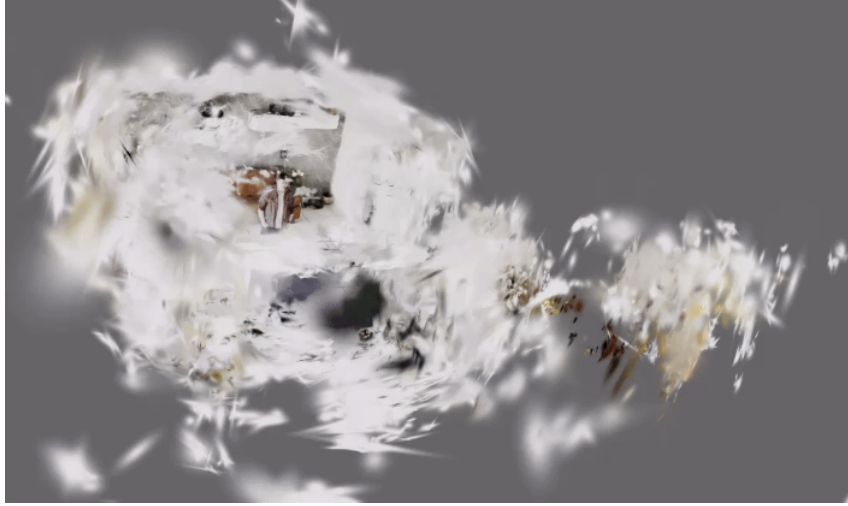


Figure 2.1: Example of a 3D Gaussian Splatting reconstruction of an environment. Taken from [2]. The environment is represented using optimized 3D Gaussian primitives, with visible reconstruction artifacts in regions with limited or uncertain scene information. These artifacts are more pronounced for viewpoints for which the 3DGS model was not optimized.

A single 3D Gaussian can be described by its center position, denoted by μ , and its covariance matrix, denoted by Σ . The Gaussian function at a 3D point x is defined as

$$G_i(p) = \sigma(\alpha_i) \exp \left(-\frac{1}{2} (p - \mu_i)^T \Sigma_i^{-1} (p - \mu_i) \right). \quad (2.1)$$

In this equation, μ determines the center of the Gaussian in 3D space, while Σ determines its size, shape, and orientation. A Gaussian with equal scale in all directions is spherical, while a Gaussian with different scales in different directions becomes elongated. This makes it possible to approximate surfaces and scene details more accurately.

The covariance matrix can be expressed using a rotation component R and a scale component S :

$$\Sigma = R S S^T R^T. \quad (2.2)$$

Here, S controls the size of the Gaussian along its local axes, while R controls its orientation in space. This allows each Gaussian to be stretched and rotated to match the structure of the scene.

A typical 3D Gaussian Splatting pipeline starts from a set of images of the target scene. From these images, camera poses and an initial point cloud can be estimated using Structure-from-Motion methods, such as COLMAP [16]. The point cloud is then used to initialize the Gaussian positions. During optimization, the position, scale, orientation, opacity, and color of the Gaussians are adjusted so that rendered views match the original input images.

Compared with manually modeling the entire factory as a mesh-based environment, Gaussian Splatting can preserve photorealistic visual details from the real scene while reducing the amount of manual modeling work. This makes it suitable for synthetic dataset generation, where images need to be rendered from multiple camera viewpoints.

However, without additional modifications, 3D Gaussian Splatting is primarily suited for scenes captured under static lighting conditions. The appearance encoded by the Gaussian representation is optimized for the illumination present during capture and does not inherently support changes in scene lighting [17]. Since this thesis also requires dynamic actors, such as pedestrians, forklifts, and industrial vehicles, these must be added separately as traditional 3D assets. Gaussian Splatting therefore provides the static factory environment, while additional simulation components are used to create dynamic scenarios.

2.4 Data Augmentation

Data augmentation is a machine-learning technique that increases the effective size and diversity of a training dataset by applying transformations to existing samples. This is achieved by taking each image sample and applying various transformations that alter the sample while preserving its original subject. The purpose of this process is to improve the generalization of the model trained on the augmented dataset. By increasing dataset diversity while preserving task-relevant content, augmentation can improve model generalisation [3]. We use the following augmentations from the standard Ultralytics library [3], which are described in detail in the subsequent sections:

1. Color and Illumination Augmentations
2. Geometric Augmentations

2.4.1 Color and Illumination Augmentations

This is a basic augmentation technique in which hue, saturation, and brightness are varied to simulate different lighting conditions and colour variations. As a result, the model may become less dependent on specific lighting and colour conditions and more robust to colour-space and brightness shifts. It typically involves the transformations described in the following sections:

2.4.1.1 Hue Adjustment

It shifts colors while preserving their relationships. The `hsv-h` hyperparameter defines the shift magnitude, with the final adjustment randomly chosen between `-hsv-h`

and hsv-h. Figure 2.2 shows the hue transformation at different parameter values, with 0.0 representing the reference image.

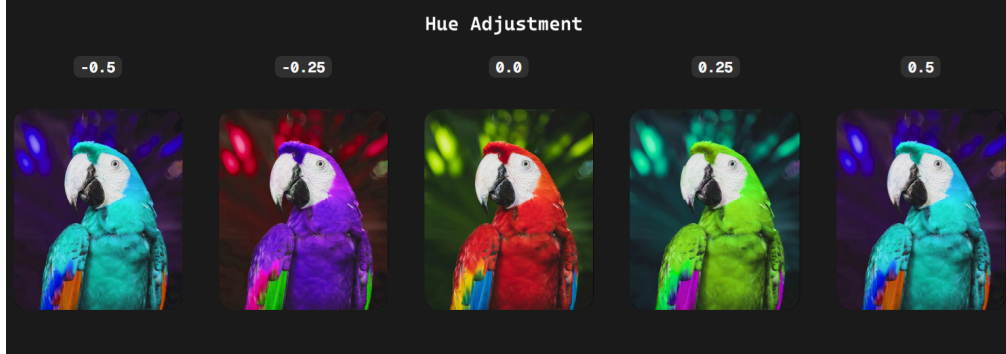


Figure 2.2: Examples of hue augmentations taken from [3]

2.4.1.2 Saturation Adjustment

It modifies the intensity of colors in the given image. The hsv-s hyperparameter defines the shift magnitude, with the final adjustment randomly chosen between -hsv-s and hsv-s. Figure 2.3 shows the saturation transformation at different intervals.

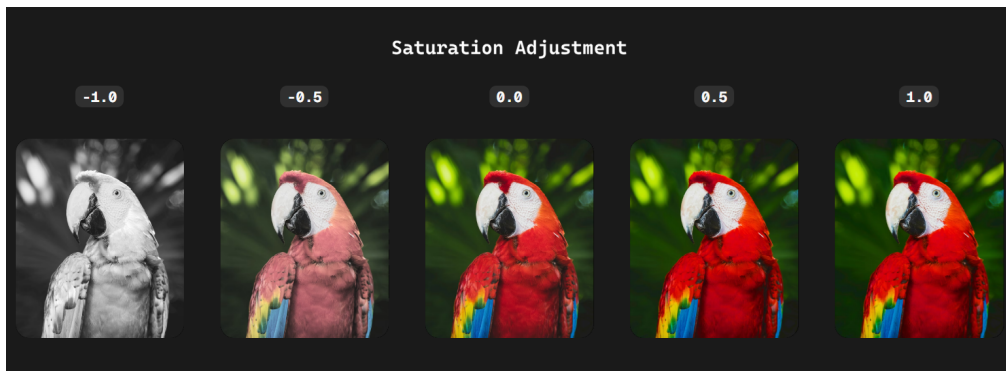


Figure 2.3: Examples of saturation augmentations taken from [3]

2.4.1.3 Brightness Adjustment

It modifies the brightness of the image. The hsv-v hyperparameter defines the shift magnitude, with the final adjustment randomly chosen between -hsv-v and hsv-v. Figure 2.4 shows the brightness transformation applied at different intervals.

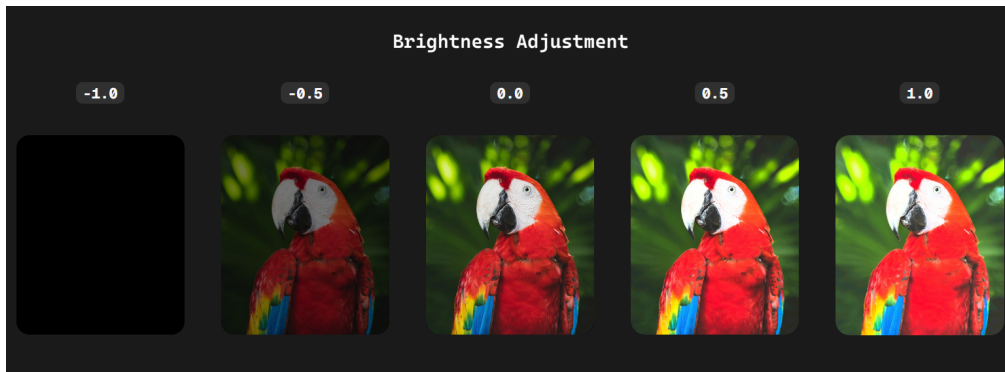


Figure 2.4: Examples of brightness augmentations taken from [3]

2.4.2 Geometric Transformations

These include geometric transformations such as scaling, flipping, and translation in two-dimensional image coordinates. The following sections describe the transformations used during model training, as implemented in the Ultralytics library:

2.4.2.1 Translation Augmentation

Translation augmentation shifts an image horizontally or vertically by a specified amount. This reduces spatial bias and thereby improves generalization. The translate hyperparameter defines the shift magnitude, with the final adjustment randomly chosen independently for each axis within the range from $-translate$ to $+translate$ as shown in Figure 2.5.

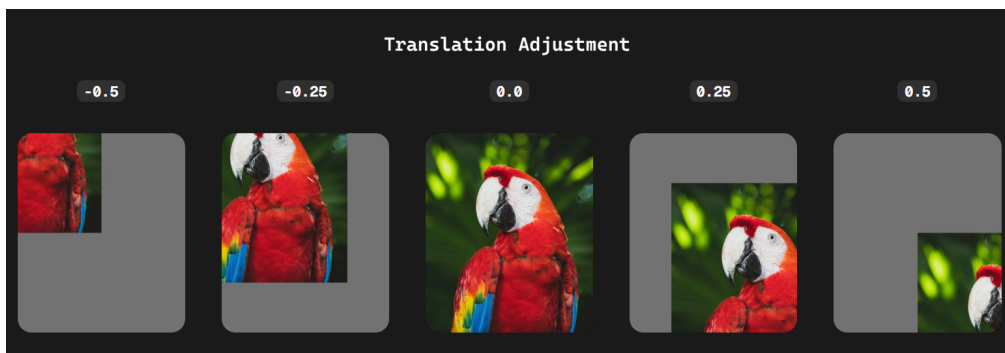


Figure 2.5: Examples of translation augmentations taken from [3]

2.4.2.2 Scale Augmentation

Random scaling encourages scale invariance, making object detection less sensitive to variations in object size. It resizes images by a random factor within the specified range as shown in Figure 2.6.

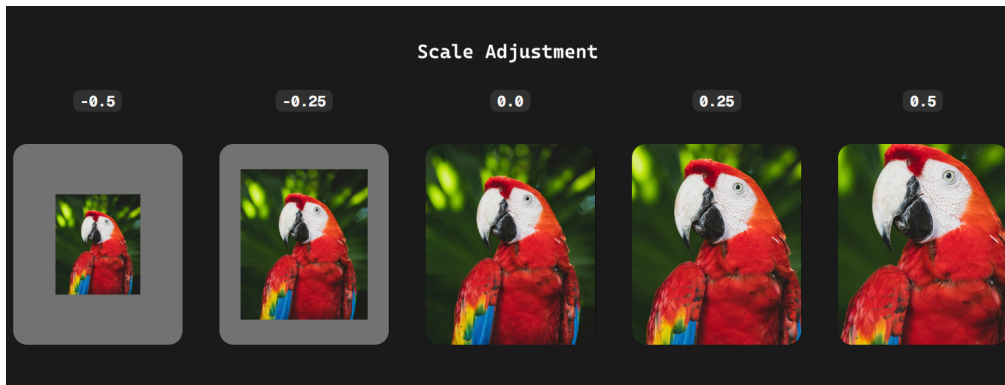


Figure 2.6: Examples of scale augmentations taken from [3]

2.4.2.3 Horizontal Flip Augmentation

Horizontal flip augmentation mirrors the image along the x-axis. This transformation helps the model generalize to objects appearing in mirrored orientations. The `fliplr` hyperparameter defines the probability of applying the transformation as shown in Figure 2.7.

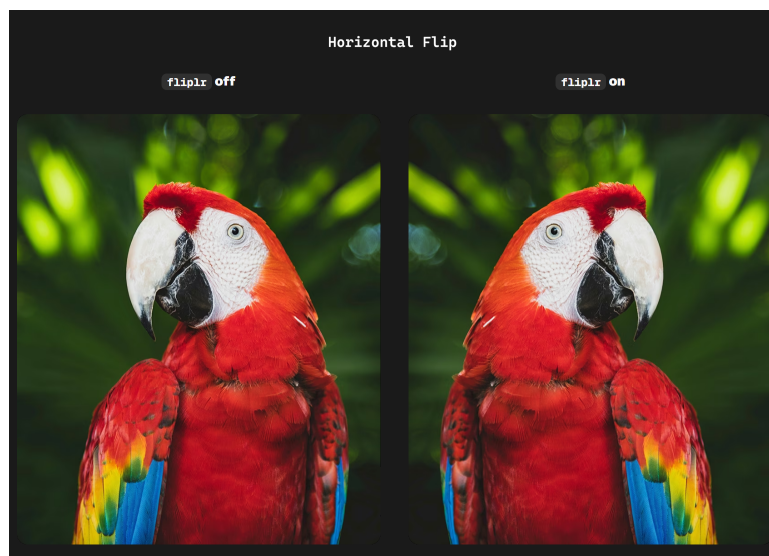


Figure 2.7: Example of Horizontal Flipping Augmentation. Taken from [3]

2.4.2.4 Mosaic Augmentation

Mosaic transformation combines four different training samples to construct a composite image. This augmentation can improve small-object detection and contextual learning, particularly when training data are limited. The `mosaic` hyperparameter defines the probability of applying the transformation as shown in Figure 2.8.

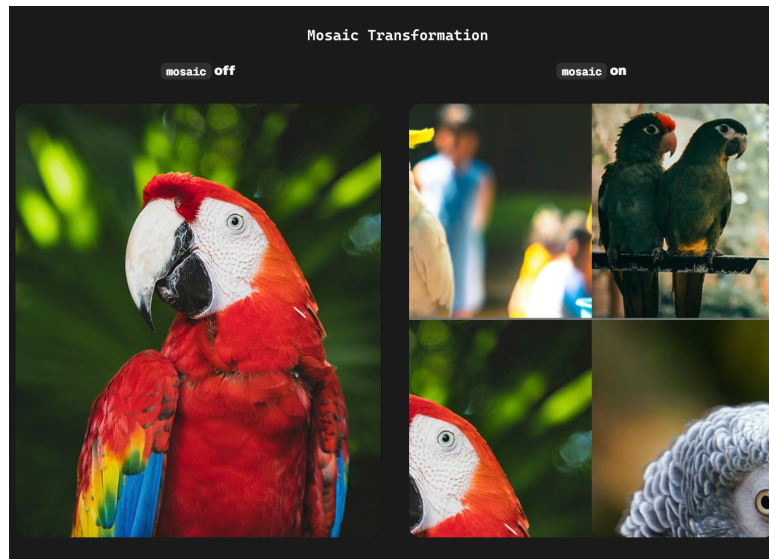


Figure 2.8: Example of Mosaic Augmentation. Useful for object detection and semantic segmentation. Taken from [3]

3

System Procedures

To construct and evaluate the dataset, a Gaussian-Splatting-based simulation environment was required. The environment needed to represent the real-world factory setting while also allowing images to be captured from multiple camera viewpoints. In addition, suitable machine learning models had to be selected in order to evaluate the generated data. To compare the performance of these models on synthetic and real-world data, a separate test dataset containing images from the corresponding real factory environment was also required.

To make the generated dataset useful for several perception tasks, such as object detection and 3D object localization, the dataset generation process was designed with the goal of producing 3D bounding boxes for each relevant object in the scene.

3.1 Data Gathering Procedure

With the environment defined, a procedure for image collection was required. Not every approach was viable because of limitations associated with Gaussian Splatting in Unreal Engine.

3.1.1 General Approach

The main problem that had to be solved was how to collect images from multiple views at the same point in time. Solving this problem would support several computer-vision applications, such as bird’s-eye view position estimation and 3D reconstruction. A number of approaches were evaluated, and the approach used in this study was selected for its simplicity and scalability.

For this approach, as shown in Figure 3.1, one camera was placed in the simulation. During image collection, the simulation is paused while this camera is sequentially moved to each configured camera position to capture an image. This preserves the scene state across all viewpoints.

The main drawbacks of this approach were increased execution times and the possibility of race conditions between camera movement, rendering, and image capture. Although synchronization measures were implemented to reduce these issues, they further slowed the simulation. Additional modifications to the environment were also required to ensure sufficient variation between collected images.

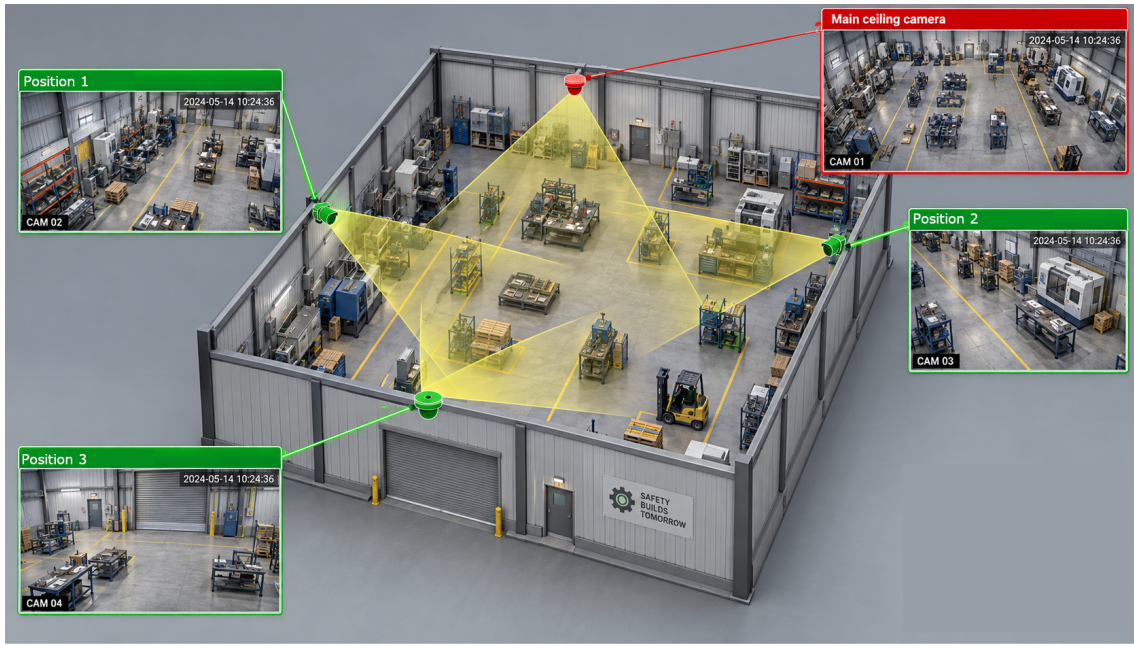


Figure 3.1: Diagram of the main approach used for this study. Generated with ChatGPT with some modifications

3.1.2 The Main Synthetic Image Generation Procedure

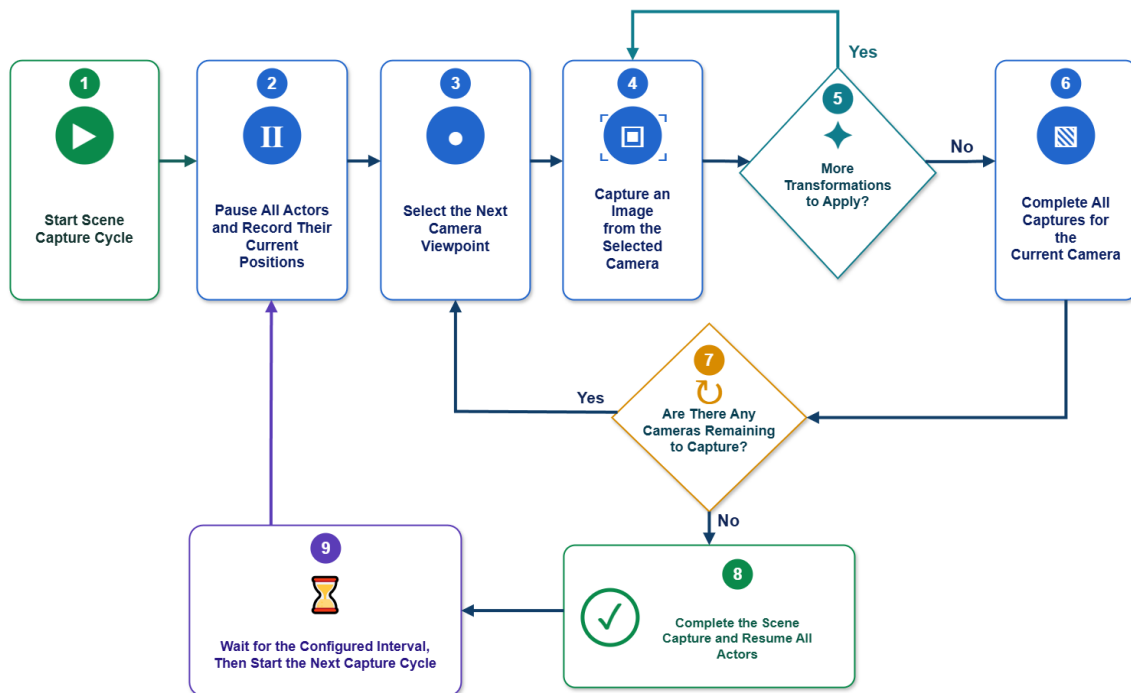


Figure 3.2: Synthetic image generation system overview

The resulting procedure, which accounts for these issues, is summarised in Algorithm 1:

Algorithm 1 Synthetic Scene Image Capture Procedure

```

1: while synthetic data generation is active do
2:   Pause all actors in the scene
3:   Record the current position and state of each actor
4:   for each camera  $c$  in the set of camera viewpoints  $\mathcal{C}$  do
5:     Select camera  $c$ 
6:     Capture and save the base image from camera  $c$ 
7:     for each transformation  $t$  in the transformation set  $\mathcal{T}$  do
8:       Apply transformation  $t$  to the scene
9:       Capture and save an image from camera  $c$ 
10:    end for
11:    Mark all required images for camera  $c$  as captured
12:  end for
13:  Resume all actors in the scene
14:  Wait for the configured capture interval
15: end while

```

The procedure above does not take into account the measures taken to avoid the race condition.

3.1.3 Moving Objects in the Simulation: Actors

To create variation between simulation states, moving actors such as pedestrians and small vehicles were added. The 3D models used for these two classes were provided by Ahmed et al. [11] as part of their bachelor's project. However, there were some limitations with these models. One limitation was that only two classes were represented by these 3D models. Another limitation was that there was only one model for each class, which limited actor diversity.

3.1.4 Actor Movement Procedure

Since the simulation operates in real time, each moving actor required an algorithm that governs its movement in a realistic manner. At its core, the movement procedure can be summarized as spawning an actor at one location and directing it toward another location within the environment. Upon reaching its destination, subsequent movement can be determined according to the defined actor behavior.

For human-like actors, an additional action was introduced to increase the diversity of the generated scenes. This action resembles bending down to pick up an object from the ground, providing greater variation in both actor poses and interactions within the simulated environment.

Algorithm 2 Moving Actor Behaviour

```
1: procedure ACTORBEHAVIOUR
2:   Spawn actor
3:   Select random action: move or pick up
4:   if action is pick up then
5:     Select random pick-up target
6:     Move to pick-up target
7:     Perform pick-up action
8:     Select random movement target
9:     Move to movement target
10:  else
11:    Select random movement target
12:    Move to movement target
13:  end if
14:  Despawn actor
15: end procedure
```

In addition, the system included “pause movement” and “restart movement” functions. The pause function interrupted the current movement and suspended all movement logic, causing the actors to remain stationary. When the “restart movement” function was activated, a new movement target was selected and normal actor behaviour resumed.

3.1.5 Actor Spawning Logic

The spawning procedure is initiated when the environment contains no active actors. At the beginning of the procedure, a predefined spawn score is assigned, which represents the available budget for actor generation. Each actor type is associated with a specific spawn cost. During each iteration, an actor type is randomly selected from the set of actors. The selected actor is spawned and its corresponding cost is deducted from the remaining score. This process is repeated until the spawn score is exhausted or no valid actor type can be selected. The remaining spawn score may become negative after the final actor is spawned. Consequently, the spawn score controls the overall composition and number of actors while still allowing variation through randomized actor selection.

3.2 Cut-and-Paste augmentations

For experiments involving the transfer of backgrounds and objects between images, separate procedures were developed for objects extracted from synthetic and real images.

3.2.1 Procedure with Simulated objects

This procedure assumes that the extracted objects follow the same file and directory structure as the standard dataset used in this project. The only difference is that

Algorithm 3 Score-Based Random Actor Spawning**Require:** Initial spawn score S , set of actor types A **Ensure:** Actors are spawned until the spawn score is exhausted

```

1: if environment is empty then
2:    $s \leftarrow S$ 
3:   while  $s > 0$  do
4:      $a \leftarrow \text{RANDOMCHOICE}(A)$ 
5:      $\text{SPAWN}(a)$ 
6:      $s \leftarrow s - \text{cost}(a)$ 
7:   end while
8: end if

```

the backgrounds have been removed from the images. Object extraction is assumed to have been performed before this procedure.

Given these initial requirements, the procedure consists primarily of replacing the background in all images with the desired background. For each camera, a background with a similar spatial configuration is applied consistently to all images captured from that camera. This process is then repeated for all cameras.

The result is a new dataset containing simulated objects placed against a different background.



Figure 3.3: Cut-and-Paste procedure for simulated objects.

3.2.2 Procedure with Real Objects

This procedure, detailed in Algorithm 4, was designed to extract foreground objects from existing camera images and place them at new positions on a different background. For each camera, the method requires an empty reference background, a new background, a drivable-region mask, input images, and input annotations. Object extraction is performed by comparing each source image with the corresponding empty background. Since the camera position is assumed to be fixed, unchanged parts of the scene should have similar pixel values, while vehicles or other foreground objects produce larger differences. The difference image is converted into a binary foreground mask using a threshold. The resulting mask is cleaned using morphological operations. Morphological operations are image-processing techniques used to change object shapes by shrinking, expanding, removing small noise, or filling small gaps[18]. Several additional checks are applied before an extracted object is

accepted; for example, objects that are too small or partially outside the image are rejected. Additionally, a manual review is used to reject poor-quality extractions that pass the automated checks. Accepted objects are stored together with their class ID and binary mask. During synthetic image generation, objects are randomly selected from the accepted object pool and randomly resized. A drivable-region mask determines where objects are allowed to appear. This mask is slightly reduced around its edges to introduce a safety margin. A placement is accepted only if all foreground pixels of the object lie inside the valid region and the overlap with previously placed objects remains below a specified limit. Once a valid position has been found, the object is blended into the new background. The mask boundary can be slightly blurred to reduce sharp Cut-and-Paste edges and make the generated image appear more natural. Finally, the annotation is recalculated from the resized and relocated foreground mask. This ensures that the generated bounding box corresponds to the object's actual position and size in the synthetic image rather than to its original source annotation. The generated image and its corresponding label file are then saved as a new training sample.

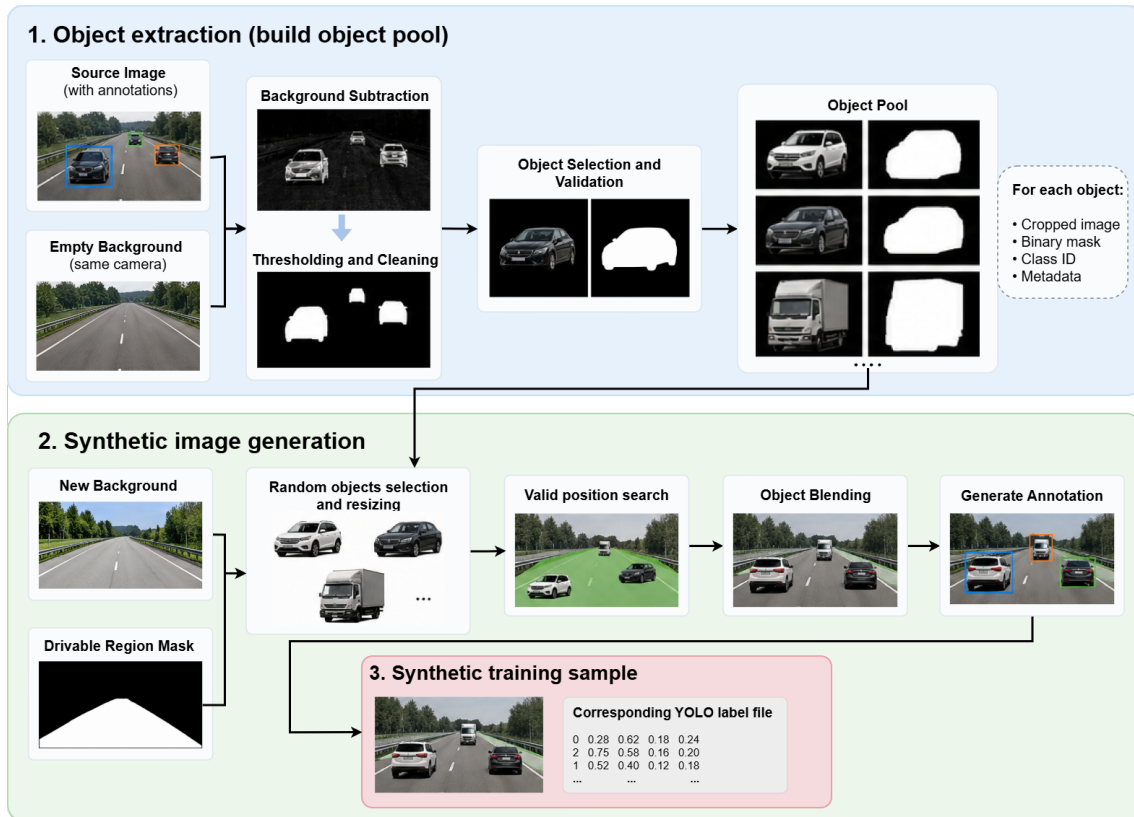


Figure 3.4: High-level overview of Cut-and-Paste dataset generation

Algorithm 4 Cut-and-Paste Image Generation

Require: Source images, labels, original background, new background, drivable-area mask

Ensure: Synthetic images with corresponding labels

```
1: for all cameras do
2:   Load the original background
3:   Initialize an empty object pool
4:   for all source images do
5:     Load the corresponding annotations
6:     for all annotated objects do
7:       Compare the source image with the empty background
8:       Create a binary foreground mask
9:       Clean the mask and select the region corresponding to the annotation
10:      Reject incomplete or invalid objects
11:      Store accepted objects and their masks in the object pool
12:    end for
13:  end for
14:  Optionally review and manually accept or reject extracted objects
15:  Load the new background and prepare the valid drivable region
16:  for  $i = 1$  to number of requested images do
17:    Copy the new background
18:    Randomly select objects from the object pool
19:    for all selected objects do
20:      Randomly resize the object
21:      Search for a valid position inside the drivable region
22:      Reject positions with excessive overlap with previously placed objects
23:      if a valid position is found then
24:        Blend the object into the background
25:        Calculate a new bounding box from the placed object mask
26:        Add the corresponding annotation
27:      end if
28:    end for
29:    Save the generated image and label file
30:  end for
31: end for
```

4

Implementation

4.1 Testing Dataset

To validate the synthetic dataset, a real-world image dataset was required. Images from a set of cameras installed in a factory environment were therefore provided for this purpose. Because the test dataset represented a factory environment, the Gaussian Splat simulation environment was designed to resemble the same setting. This dataset also contained pedestrians at different positions as well as small vehicles such as forklifts.

4.1.1 Annotation Software

The provided dataset, however, contained only images from the specified cameras and did not include annotations for the object classes present. Therefore, a dedicated annotation tool was used. This tool was specifically designed to annotate 3D bounding boxes in the Volvo factory environment, with a predefined free-space boundary based on the real factory layout.

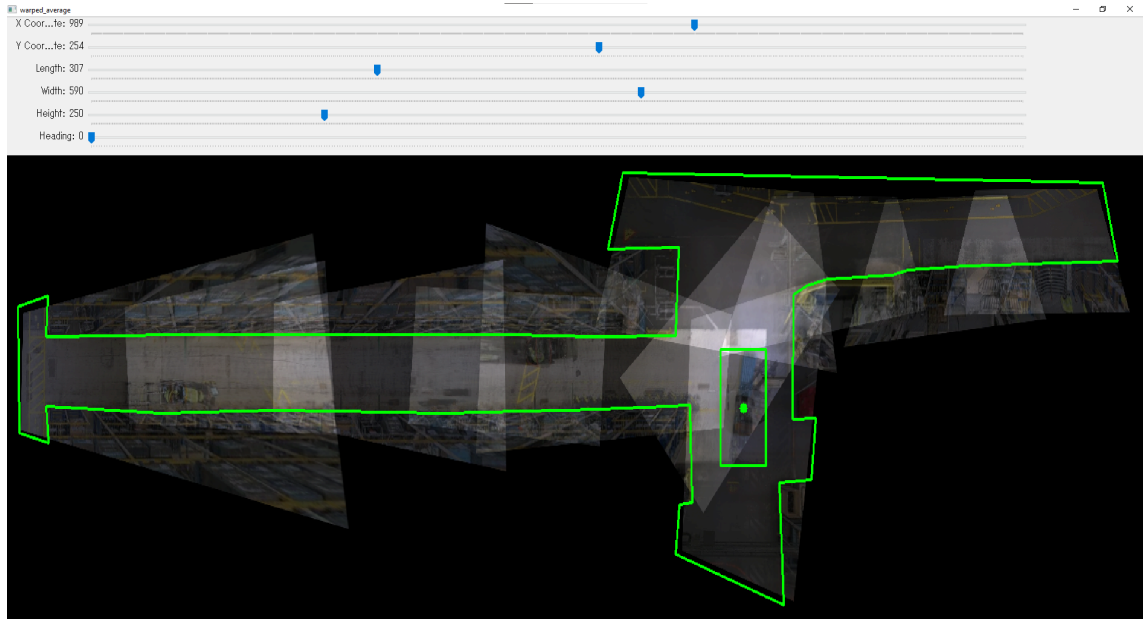


Figure 4.1: Selection of the bounding-box location and size in a bird's-eye view.

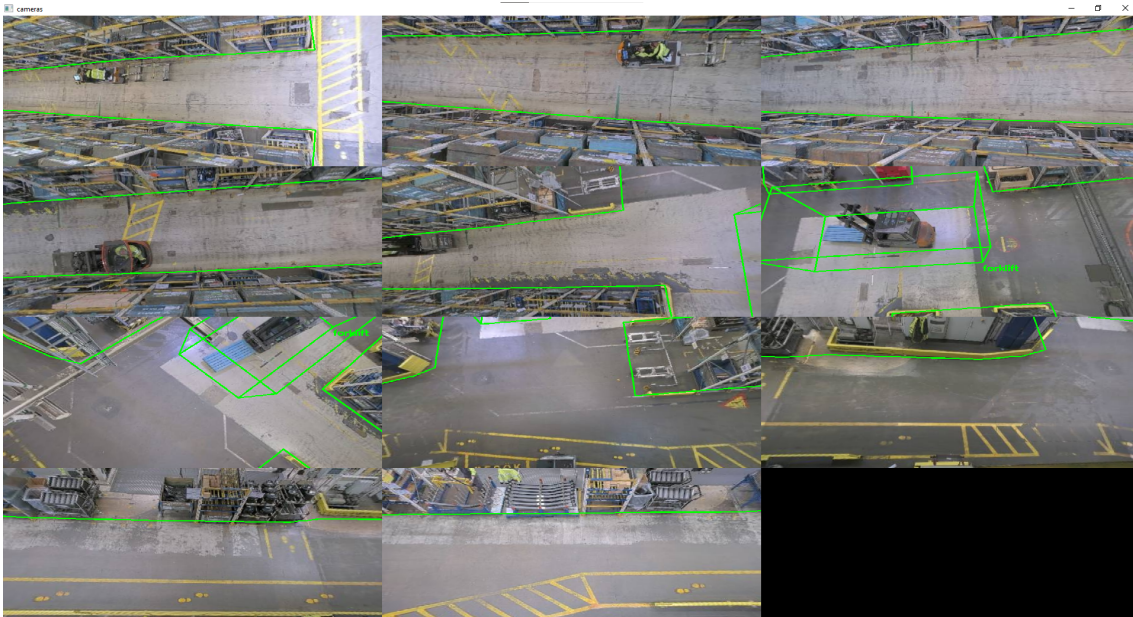


Figure 4.2: A preview of the bounding box can be seen in this window.

4.1.2 Testing Dataset Statistics

After the test-set annotations were summarised, descriptive statistics were computed to examine the class distribution and other dataset characteristics.

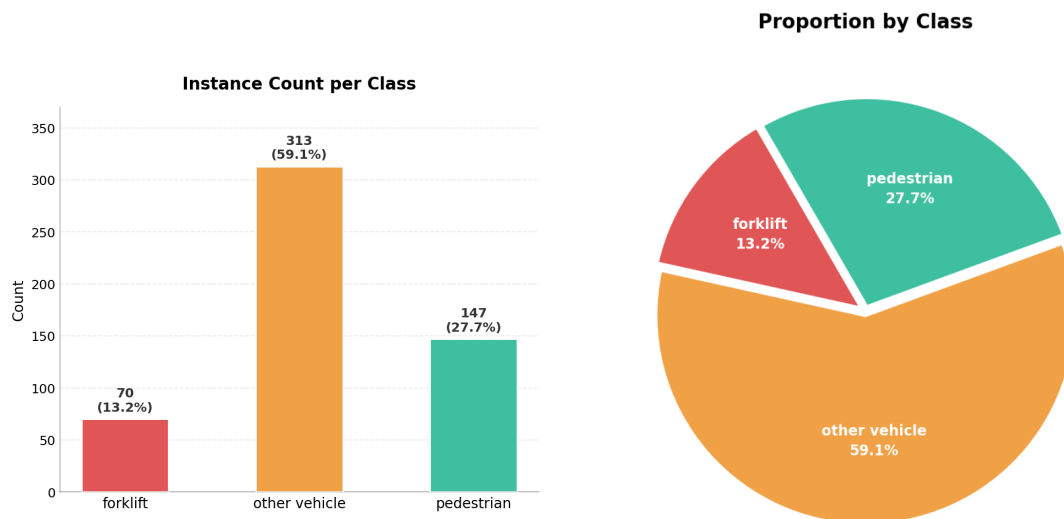


Figure 4.3: Instance distribution between classes

One notable aspect of the test dataset was that “other vehicle” was the most common class; this category included vehicles that were not classified as forklifts. Consequently, object appearance within this class varied substantially because it covered a broad range of vehicle types.

Summary Statistics Table

Class	Count	L mean $\pm$ std	W mean $\pm$ std	H mean $\pm$ std	Vol mean $\pm$ std
forklift	70	2.74 $\pm$ 0.69	2.24 $\pm$ 0.95	2.48 $\pm$ 0.53	14.31 $\pm$ 5.71
other vehicle	313	1.73 $\pm$ 0.45	1.37 $\pm$ 0.49	1.29 $\pm$ 0.34	3.28 $\pm$ 2.44
pedestrian	147	0.67 $\pm$ 0.17	0.66 $\pm$ 0.16	1.56 $\pm$ 0.29	0.73 $\pm$ 0.41
Total	530				

Figure 4.4: Summary of the overall statistics

4.1.3 Unreal Engine

Unreal Engine is a real-time 3D engine widely used for creating advanced simulations and virtual environments. Developed by Epic Games, it provides realistic physics, lighting, and environmental interactions, making it useful for applications such as robotics, autonomous driving, training systems, and digital twins. Its ability to render complex scenes in real time allows developers to test and simulate scenarios efficiently in a controlled virtual environment[19].

Unreal Engine, while powerful, does not natively support 3D Gaussian Splatting. Unreal Engine’s plugin system can be used to add the required functionality.

4.1.4 XGrids

XGRIDS provides camera-and-LiDAR systems for 3D environment capture. The systems combine high-resolution cameras, LiDAR sensors, and SLAM-based tracking to simultaneously capture visual and spatial information, enabling the reconstruction of detailed photorealistic scenes in real time. Through the XGRIDS plugin for Unreal Engine, these reconstructed environments can be imported directly into virtual simulations and interactive applications [20]. In addition to the Gaussian Splat reconstruction created by Ahmed et al. [11], a new Gaussian Splat environment was captured using XGRIDS camera technology. Its proprietary file format required the corresponding Unreal Engine plugin.

4.2 Synthetic Actor Generation Using Generative Models

To avoid manual effort and potentially improve the diversity of actors within the simulated factory environment, we adopted a two-step generative approach that combines image generation using *Sora* and 3D model generation using *Hunyuan3D*. These models were used to create a wide range of synthetic actor models, including human workers and industrial vehicles.

The generated 3D actors are then placed within a Gaussian Splatting reconstruction of the factory environment in Unreal Engine. Each actor instance was assigned randomized attributes, including:

- Spatial position and orientation

- Motion trajectories
- Interaction with scene elements

4.2.1 Motivation

Traditional simulation pipelines rely on manually designed 3D assets, which are often limited in variability and require significant manual effort to scale. To address this limitation, generative models were used to create novel actor instances with minimal manual effort.

The use of generative models enables scalable dataset creation by allowing new actor variations to be synthesized on demand. This approach avoids reliance on fixed asset libraries and supports continuous expansion of the dataset.

In contrast to traditional methods, the proposed pipeline introduces diversity along multiple axes:

- Visual appearance
- Geometry and structure
- Contextual placement within the scene

4.2.2 Actor Appearance Generation with Sora

Sora is a generative image model developed by OpenAI [21]. It is used in our experiments to generate a set of controlled and diverse visual representations for models based on text prompts. We generated visual appearances for human workers, forklifts, and other vehicles in the factory. The prompts were designed to capture variations in human actors as well as vehicles inside factory such as:

- Clothing (e.g., safety vests, uniforms, casual wear)
- Vehicle types and variations.
- Colour, contrast, and hue
- Body types and demographic diversity.
- Accessories and cargo (e.g., helmets, tools, boxes).
- Signs of wear, such as scratches and dust on vehicles

The generated outputs provided high variability in appearance, enabling the creation of a broad distribution of human worker profiles. To ensure relevance to the industrial setting, prompts were constrained to factory-specific contexts. For example, prompts describe attributes such as helmet color, type of safety jacket, and other work-related characteristics.

4.2.3 3D Asset Generation with Hunyuan3D

While Sora was used to generate visual appearance, Hunyuan3D, an open-source generative 3D model, was used to produce corresponding 3D assets suitable for integration into the simulation pipeline.

Hunyuan3D enabled the generation of textured 3D meshes from image or text inputs. In this work, it was used to generate `.obj` models for corresponding actors, including full 3D geometry and associated textures.



Figure 4.5: Examples of appearances generated using Sora for human and vehicle actors in the factory.

The resulting meshes were further processed to ensure compatibility with the rendering framework. This processing included scale normalisation, coordinate-system alignment, and optimisation for real-time rendering.

4.3 Simulation Construction and Main Procedure

With all components required to construct the simulation defined, the overall construction methodology was as follows:

The first step was to import the 3DGS model into Unreal Engine. The 3DGS model used in all experiments was provided by Björkman et al. [1]. The 3DGS representation does not provide collision geometry for actors in the simulation. To address this limitation, a simplified mesh representation of the environment was created. This representation primarily consisted of walls and a floor, as its purpose was to provide the necessary collision and navigation geometry rather than to reproduce the visual

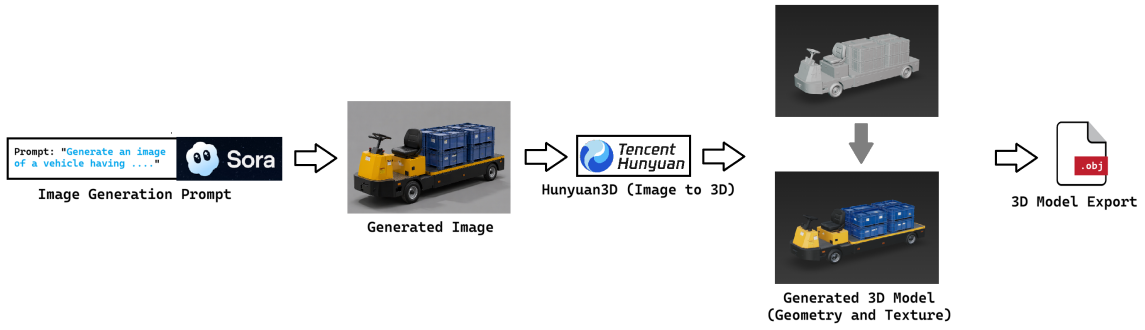


Figure 4.6: Generative assets generation pipeline using Hunyuan3D and Sora

appearance of the environment. The navigation geometry defined the areas in which the actors were able to move. Since the mesh environment served only a functional purpose, it was typically not visible during runtime.

Cameras could then be positioned throughout the environment at different locations and viewing angles, depending on the requirements of the user. Before the synthetic image generation procedure could be executed, the actors used in the simulation also had to be defined. Making use of the previously described spawning and pausing logic, actor management within the simulation could be handled automatically. The remaining step was therefore to import a suitable collection of actor models.

4.3.1 Actor Model Animations

Most non-human actor models, such as forklifts and other vehicle types, did not require specific animations for the purposes of this project. Human models, however, required at least walking and idle animations to allow them to behave appropriately within the simulated environment.

As the human models used in this project were generated using Hunyuan3D, animations were not provided alongside the generated models. To address this, Mixamo, Adobe’s online platform for automatically rigging 3D characters and applying pre-defined motion-captured animations, was used to rig and animate the models [22]. By uploading each human model to Mixamo, the required animations could be generated for each human model. To integrate the animations with the human models, the animation data first had to be processed in Blender, a free and open-source 3D creation suite that supports tasks such as 3D modeling, animation, rendering, simulation, compositing, and video editing [23]. Following this processing step, the animated human models could be imported into Unreal Engine. Once the environment, camera setup, actor models, and corresponding animations had been configured, the synthetic image-generation pipeline could be executed.

With the environment, cameras, actor models, and corresponding animations configured, the synthetic image generation procedure can then be executed.

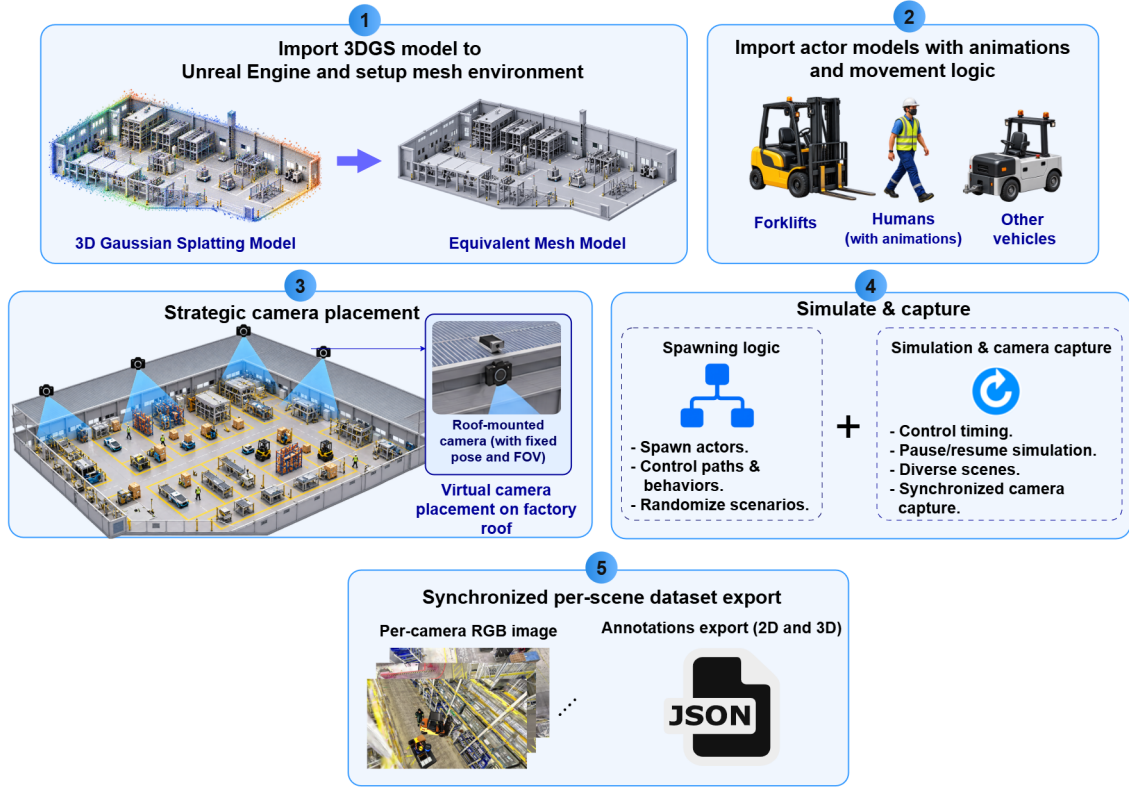


Figure 4.7: High-level overview of the simulation-environment construction process.

4.4 Resultant Data

The output of the main procedure was the resulting dataset, which consisted of the collected images and a text file containing the positions and rotations of all actors in the scene. Specifically, each data point consisted of one image from each camera view together with the x-, y-, and z-coordinates and yaw rotation of all actors. An example of the output generated by Unreal Engine is shown below:

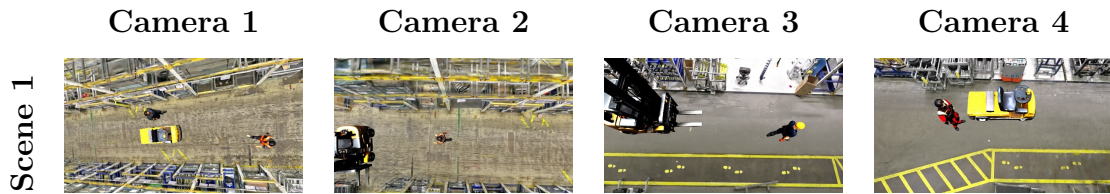


Figure 4.8: Synthetic images generated in the UE5 simulation from four different camera viewpoints for a random scene

Additionally, the exported labels for a scene look like this:

```
Human0: (-2.807038,-38.312116,6.198836, Yaw: -90.222618), HumanBox0: (X
=0.272 Y=0.272 Z=0.758)
Human1: (-1.400983,-27.881956,6.197894, Yaw: -93.558342), HumanBox1: (X
=0.272 Y=0.272 Z=0.758)
Forklift0: (-2.44269,-29.325548,6.250693, Yaw: -149.028687), ForktBox0:
(1.240805,0.46537,0.73117)
Vehicles0: (-0.599149,-30.687535,6.250433, Yaw: 86.861656), VehitBox0:
(0.9977,0.43805,0.557124)
```

Each label contains x-, y-, and z- coordinates together with x-, y-, and z-dimensions that define a box around the actor.

4.5 Processing the Data

Although the data had been collected, it was not yet suitable for use in the intended machine-learning pipeline. This was because all position and rotation data used Unreal Engine conventions, which differed from those required by the machine-learning models. Unreal Engine uses a left-handed coordinate system with the z-axis pointing upward. Furthermore, the rotation directions for roll, pitch, and yaw are clockwise, clockwise, and counterclockwise, respectively. A conversion script was therefore developed to transform Unreal Engine coordinates into the conventions required by the machine-learning model.

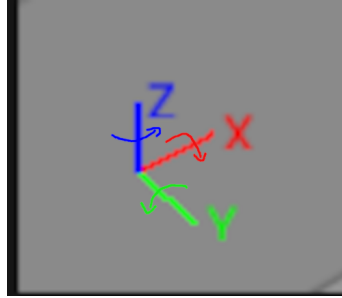


Figure 4.9: An illustration of the Unreal Conventions. Taken directly from Unreal Engine application with added arrows for clarification.

4.5.1 Camera Calibration Conversion

For each camera, the Unreal Engine position is defined as

$$\mathbf{c}_{UE} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}.$$

The pitch, yaw and roll angles are converted to radians with the sign convention

$$\theta_p = -P, \quad \theta_y = Y, \quad \theta_r = -R.$$

The rotation matrices are

$$R_x(\theta_r) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_r & -\sin \theta_r \\ 0 & \sin \theta_r & \cos \theta_r \end{bmatrix},$$

$$R_y(\theta_p) = \begin{bmatrix} \cos \theta_p & 0 & \sin \theta_p \\ 0 & 1 & 0 \\ -\sin \theta_p & 0 & \cos \theta_p \end{bmatrix},$$

$$R_z(\theta_y) = \begin{bmatrix} \cos \theta_y & -\sin \theta_y & 0 \\ \sin \theta_y & \cos \theta_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The Unreal Engine camera-to-world rotation is computed as

$$R_{cw}^{UE} = R_z(\theta_y)R_y(\theta_p)R_x(\theta_r).$$

The corresponding world-to-camera rotation is

$$R_{wc}^{UE} = \left(R_{cw}^{UE}\right)^T.$$

The Unreal Engine translation vector is

$$\mathbf{t}^{UE} = -R_{wc}^{UE} \mathbf{c}_{UE}.$$

Thus, the Unreal Engine extrinsic matrix is

$$[R|t]_{UE} = \begin{bmatrix} R_{wc}^{UE} & \mathbf{t}^{UE} \end{bmatrix}.$$

To convert from Unreal Engine coordinates to OpenCV coordinates, the following conversion matrix is used:

$$C = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}.$$

The OpenCV extrinsic matrix is therefore

$$[R|t]_{CV} = C[R|t]_{UE}.$$

Finally, the camera projection matrix is computed as

$$P = K[R|t]_{CV},$$

where K is the intrinsic camera matrix.

For a homogeneous world point

$$\mathbf{X}_w = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

the projected image point is

$$\mathbf{x} = P\mathbf{X}_w = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix}.$$

The pixel coordinates are then obtained by perspective division:

$$u = \frac{u'}{w'}, \quad v = \frac{v'}{w'}.$$

A valid projection requires positive depth:

$$w' > 0.$$



Figure 4.10: Comparison of the original image (left) and the same image with 3D bounding-box annotations surrounding the detected actors (right).

4.5.2 3D to 2D Bounding box Conversion

From 3D bounding boxes, several types of annotations can be derived, one of which is 2D bounding boxes. The implemented script converts existing 3D bounding-box annotations into 2D bounding-box labels for a set of calibrated cameras. Three object classes are considered: `forklift`, `other vehicle`, and `pedestrian`, which are mapped to class IDs 0, 1, and 2, respectively.

For each camera, the intrinsic matrix $\mathbf{K}$ and the extrinsic rotation and translation parameters $\mathbf{R}$ and $\mathbf{t}$ are loaded from the corresponding calibration files. The camera projection matrix is constructed as

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}. \quad (4.1)$$

For each object, the 3D bounding-box corners are first transformed from the world coordinate system into the camera coordinate system according to

$$\mathbf{X}_c = \mathbf{R}\mathbf{X} + \mathbf{t}. \quad (4.2)$$

The corresponding homogeneous 3D point $\tilde{\mathbf{X}}$ is then projected onto the image plane using

$$\tilde{\mathbf{x}} = \mathbf{P}\tilde{\mathbf{X}}, \quad (4.3)$$

where $\tilde{\mathbf{x}} = (\tilde{u}, \tilde{v}, \tilde{w})^T$. The final pixel coordinates are obtained through perspective division as

$$u = \frac{\tilde{u}}{\tilde{w}}, \quad v = \frac{\tilde{v}}{\tilde{w}}. \quad (4.4)$$

A visibility criterion is applied before generating a 2D bounding box. First, only 3D corners with positive depth in the camera coordinate system are considered,

$$Z_c > 0. \quad (4.5)$$

At least four bounding-box corners must satisfy this condition; otherwise, the object is discarded for that camera. Among the projected corners with positive depth, at least three must lie within the image boundaries for the object to be retained.

After the visibility test, an axis-aligned 2D bounding box is constructed using the minimum and maximum image coordinates of all projected corners with positive depth:

$$x_{\min} = \min_i u_i, \quad x_{\max} = \max_i u_i, \quad (4.6)$$

$$y_{\min} = \min_i v_i, \quad y_{\max} = \max_i v_i. \quad (4.7)$$

The resulting coordinates are clipped to the image boundaries. Bounding boxes with invalid dimensions or an area smaller than 50 px² are discarded.

Since direct projection of a 3D cuboid can result in relatively loose 2D bounding boxes, an additional artificial tightening step is applied to the **forklift** and **other vehicle** classes. The center of the bounding box is preserved while its width is reduced by 20% and its height by 10%. If the original bounding-box width and height are denoted by w_b and h_b , the adjusted dimensions are

$$w'_b = 0.8w_b, \quad h'_b = 0.9h_b. \quad (4.8)$$

The adjusted coordinates are then reconstructed around the original bounding-box center and clipped again to the image boundaries. No artificial tightening is applied to **pedestrian** bounding boxes.

Finally, each valid 2D bounding box is converted into the normalized YOLO representation

$$(c, x_c, y_c, w, h), \quad (4.9)$$

where c denotes the object class ID and

$$x_c = \frac{x_{\min} + x_{\max}}{2W}, \quad y_c = \frac{y_{\min} + y_{\max}}{2H}, \quad (4.10)$$

while the normalized bounding-box width and height are given by

$$w = \frac{x_{\max} - x_{\min}}{W}, \quad h = \frac{y_{\max} - y_{\min}}{H}, \quad (4.11)$$

where W and H denote the image width and height, respectively. The resulting annotations are stored as YOLO label files separately for each camera, together with the corresponding camera images.



Figure 4.11: Comparison of the original image (left) and the same image with 2D bounding-box annotations over the detected actors (right).

4.6 Computer Vision Model: YOLOv8

YOLO (You Only Look Once) is a family of single-stage object detection models designed to perform object localisation and classification in a single forward pass through a neural network [24]. Unlike two-stage detectors, which first generate candidate regions before classifying them, YOLO directly predicts bounding boxes and class probabilities from the input image. This design provides a favorable balance between detection accuracy and computational efficiency, making the YOLO family well suited to real-time object detection applications [25, 26].

In this thesis, **YOLOv8** is used as the object detection model to evaluate the synthetic data generated. YOLOv8 is a later generation of the YOLO framework developed by Ultralytics and introduces architectural and training improvements over earlier YOLO versions. Its detection pipeline consists of a backbone for extracting visual features, a neck for combining features at different scales, and a detection head that predicts object classes and bounding boxes. This multi-scale feature representation is particularly useful for our dataset, where objects may occur at different distances and scales within the scene.

The primary role of YOLOv8 in this work is to provide a consistent object-detection framework for evaluating the generated synthetic images and analysing the sim-to-real gap using standard metrics. In particular, we initialise YOLOv8m from a pre-trained checkpoint and fine-tune the model on our datasets so that it can exploit the pre-trained feature representation.

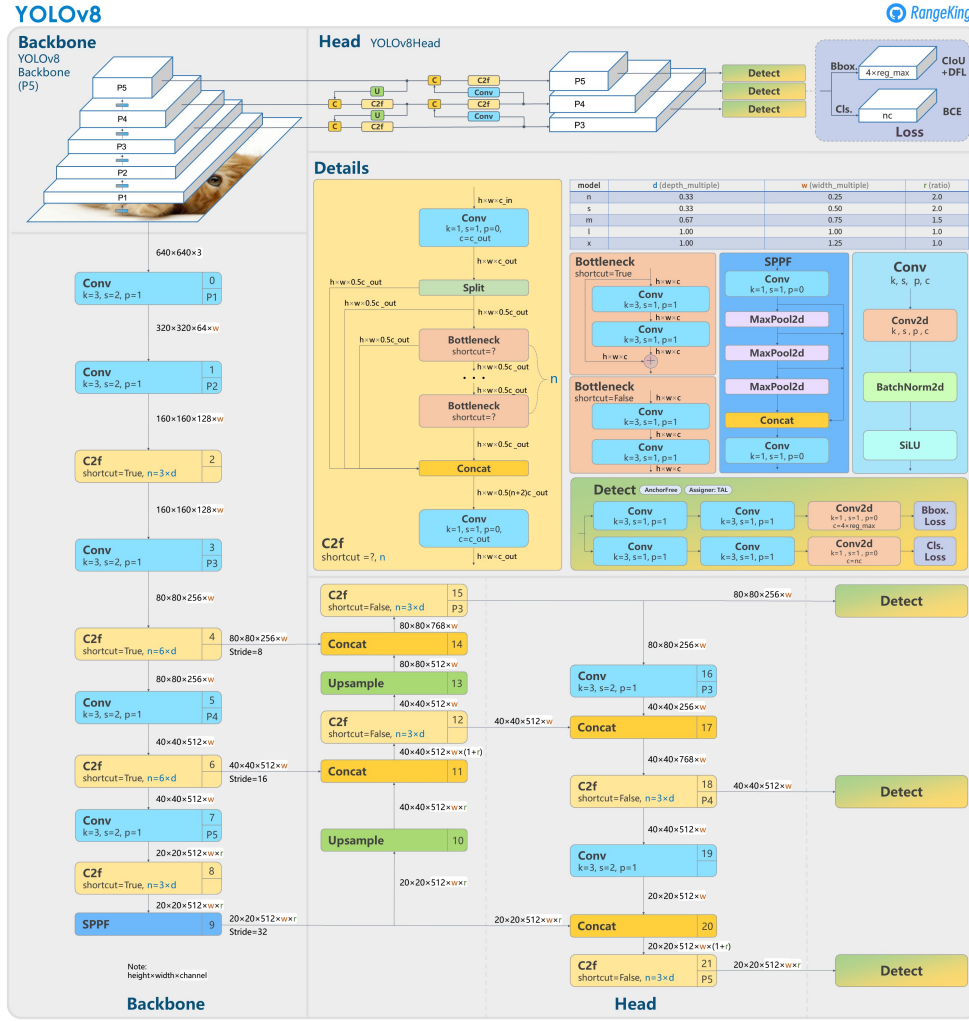


Figure 4.12: Illustration of the YOLOv8 architecture. Taken from [4]

4.6.1 Model Setup and Hyperparameters

For model training, the input, output, and a set of hyperparameters had to be defined. The model input was a 640×360 pixel image. The model output was multiclass object detection with three labels: forklift, other vehicle, and pedestrian. The main hyperparameters and training settings were as follows:

- Pre-trained model - Although a model can be trained from scratch, performance can often be improved by initializing from a pre-trained model. The pre-trained model should ideally have been trained on a task similar to the target task; for example, an object-detection checkpoint is appropriate for an object-detection task. The pre-trained model used for this project was "Yolov8m.pt" which is a medium-sized variant of the YOLOv8 object-detection family. The yolov8m.pt model is pre-trained on the Microsoft COCO dataset. COCO contains 118,287 training images annotated across 80 common object categories [27, 28].
- Runtime - The training consisted of a total of 200 epochs with early stopping implemented. This meant that if there were no improvement within a number

of epochs, the training would stop early. This value was set to an integer between 25 and 35.

- **Class Reweighting** - Since there was no control over how many annotated objects of each class the synthetic dataset generates, class reweighting was applied to account for any overrepresentation. This was implemented using the variable “cls_pw”, which controls inverse-frequency weighting of the classification loss. The initial weight assigned to class (c) is proportional to

$$w_c = \left(\frac{1}{n_c} \right)^{0.5}, \quad (4.12)$$

where n_c is the number of training instances belonging to that class. The resulting class weights are normalized before being applied to the binary cross-entropy classification loss. This gives relatively greater importance to less frequent classes without applying full inverse-frequency weighting. [29]

- **Learning Rate and Optimizer** - The model used the MuSGD optimizer with a learning rate of 0.01 and a momentum of 0.9.
- **Augmentation Values** - For each data augmentation, a set value was used to determine the frequency of the data augmentation as well as how strongly the data augmentation is applied. For this training procedure, the data augmentation values were:
 - **Hue: 0.015** – Applies small random shifts to the image hue, introducing slight variations in object and background colors.
 - **Saturation: 0.7** – Varies the color intensity of the image, allowing colors to appear more or less saturated.
 - **Brightness: 0.4** – Varies the brightness of the image, generating darker and brighter versions of the training samples.
 - **Translation: 0.1** – Shifts the image horizontally and vertically by up to approximately 10% of the image dimensions.
 - **Scaling: 0.5** – Randomly changes the image scale, corresponding to approximately 0.5–1.5 times the original scale.
 - **Horizontal flip: 0.5** – Gives each training image a 50% probability of being mirrored horizontally.
 - **Mosaic: 1.0** – Applies Mosaic augmentation with full probability while active, combining four training images into one composite image.

4.7 Cut-and-Paste Implementation

For the given Cut-and-Paste synthetic image generation procedure, YOLO bounding box labels were used to extract the objects from the testing dataset. An empty background from each camera in the test dataset was used for comparison to identify and extract objects from the images. In order to minimize spatial mismatch when pasting the objects, the new backgrounds were constructed to have similar spatial construction to the old backgrounds. This was done by positioning the virtual cameras in Unreal Engine so that their viewpoints matched the original background images as closely as possible. The final Cut-and-Paste datasets each contained 1000

scenes, with one scene being one image per camera. This size was chosen to match the other datasets constructed during the project.

4.8 Visual Domain Adaptation using Generative Diffusion Models

The visual domain gap is one contributor to the sim-to-real gap, particularly in perception tasks where synthetic and real images differ because of rendering style, colour, hue, noise, and other visual factors. Benigmin et al. [30] used diffusion-based image-to-image generation to transfer visual style from real to synthetic images, demonstrating its potential as a domain-adaptation technique for reducing the sim-to-real gap.

We introduced a similar approach in which a generative image-to image translation pipeline used a reference image from the target domain together with a text prompt. The proposed pipeline took a synthetic image, a text prompt and a corresponding real image from a similar camera viewpoint as a reference. The model generated a transformed version of the synthetic image that aims to retain the scene content while adapting its visual characteristics to those of the real domain.

The transformation can be formulated as follows:

$$\hat{x}_{s \rightarrow t} = G(x_s, x_t, p), \quad (4.13)$$

where,

x_s denotes the synthetic source image.

x_t denotes a real target-domain reference image.

p denotes a natural-language instruction describing the desired transformation.

G denotes the generative image model.

The resulting image $\hat{x}_{s \rightarrow t}$ is intended to preserve the overall scene structure of x_s while making its appearance more representative of the real target domain.

An important characteristic of the proposed approach is that the transformation is not limited to low-level visual properties such as color, lighting, and hue. The generative model can also modify object appearance to better match real-world counterparts. This allows synthetic objects to acquire more realistic characteristics and may further reduce the visual discrepancy.

4. Implementation

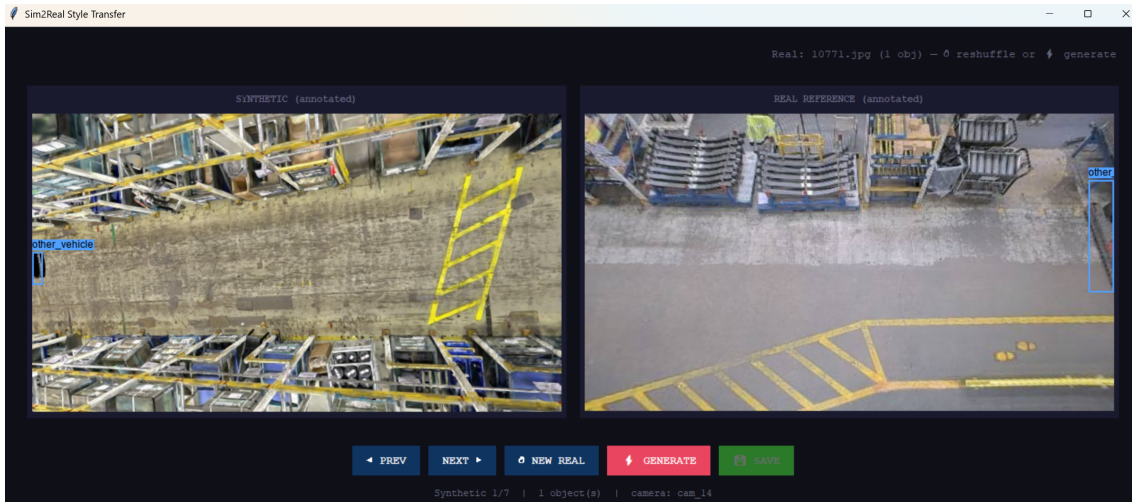


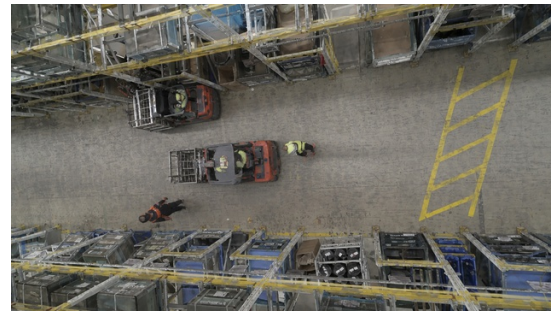
Figure 4.13: Tool developed for image-to-image translation

Figure 4.13 shows the tool developed for the proposed image-to-image translation pipeline. We specifically used OpenAI’s Images 2.0 multimodal foundation model for image-to-image translation. The left image is a synthetic image, and the right image is a reference image from the real dataset. The text prompt does not change and is therefore hard-coded as a system prompt that is not displayed in the tool. The tool processes synthetic images sequentially and outputs the corresponding style-transferred images.

The process was not fully automatic. In some cases, the generative model introduced small changes to object geometry or position and therefore, some of the generated images required manual annotation adjustment before being used for subsequent training. Nevertheless, the proposed pipeline reduced the amount of manual effort compared to creating annotations for real images entirely from scratch, while providing a practical approach for generating synthetic data with visual characteristics closer to those of the target domain.



A sample synthetic image before image-to-image translation

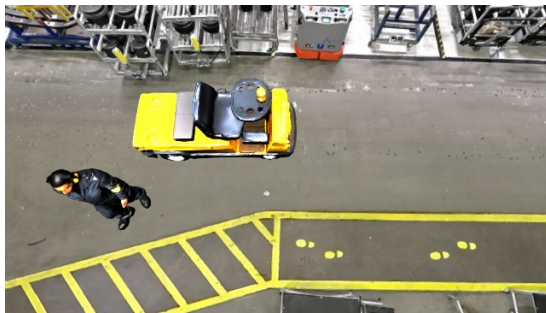


Corresponding style transferred output after image-to-image translation

Figure 4.14: Side-by-side comparison of a source image and a style-transferred image in which object appearances are adapted toward the real domain.

In Figure 4.14, the left image shows the synthetic image before translation, and the

right image shows the result after translation. As shown in the figure, in addition to changing overall style it also replaced objects in the scene with their realistic counterparts. This was particularly useful because the task was to detect these objects, and greater similarity to their real counterparts could improve transfer to the real domain. One limitation of this approach was that it sometimes modified object geometry and position, making the images geometrically inconsistent from a multi-view perspective. However, in our experience this depended on prompt instructions. If we designed prompts which allowed for replacing or modifying objects with their real counterparts it tended to modify geometry and sometimes position and poses of objects. However, when prompts explicitly restricted changes to object geometry, geometry was generally preserved and only the image style changed, as shown in Figure 4.15



A sample synthetic image before image-to-image translation



Corresponding style transferred output after image-to-image translation

Figure 4.15: Side-by-side comparison of a source image and a style-transferred image with object geometry preserved.

5

Results

This chapter presents the experimental results evaluating the synthetic dataset generated using our pipeline for object detection in a large-scale industrial environment. The synthetic datasets were generated utilizing Unreal Engine 5 and 3D Gaussian Splatting based digital twin reconstruction based on a real factory environment. An automated dataset generation pipeline was implemented in Unreal Engine 5 to render RGB images from virtual ceiling-mounted cameras similar to the arrangement in the real factory environment. The generated datasets were evaluated on an industrial object detection task using YOLOv8m as the detection backbone. The main task was to detect three object classes: *forklift*, *other vehicle*, and *pedestrian*. Finally, the real-world evaluation was performed on manually annotated real images captured by ceiling-mounted cameras in the factory to assess the transferability of models trained on synthetic data.

5.1 Evaluation Metrics

Detection performance was evaluated using the standard COCO metrics. Mean Average Precision at an Intersection over Union (IoU) threshold of 0.5 (**mAP50**) measures detection performance under a relatively relaxed localization criterion and serves as the primary evaluation metric throughout this chapter. Mean Average Precision averaged over IoU thresholds ranging from 0.5 to 0.95 in increments of 0.05 (**mAP50–95**) provides a more stringent assessment of both detection and localization performance. Precision and Recall are also reported for completeness. For evaluation on the real-world dataset, a total of 957 images distributed evenly across 11 different ceiling-mounted cameras were manually annotated, resulting in 87 annotated images per camera.

5.2 Dataset Overview

The synthetic datasets were generated in Unreal Engine 5 (UE5) using a Gaussian Splatting reconstruction of the real factory as the virtual 3D environment. Three object classes, namely *forklifts*, *other vehicles*, and *pedestrians*, were represented by 3D assets generated using generative AI techniques and then imported into the reconstructed environment for simulation.

Using this virtual environment, two synthetic datasets were generated, each consisting of 1000 scenes but employing different camera configurations of 9 and 21 cameras, respectively. Each dataset contains variations in camera configuration, scene layout,

object distribution, and background conditions for training deep-learning models, as discussed in the following sections.

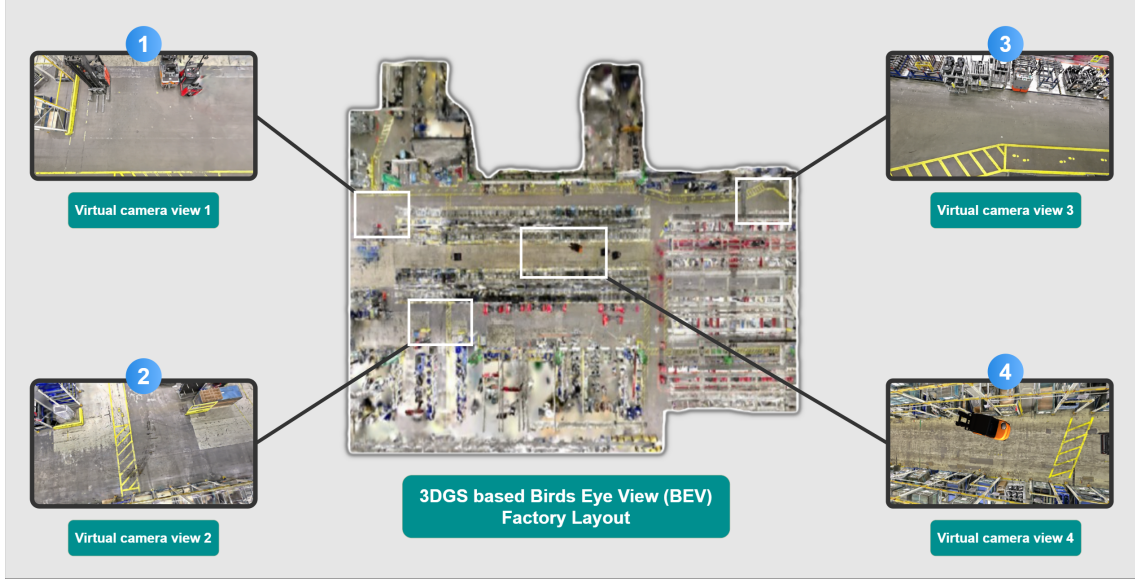


Figure 5.1: Synthetic factory layout shown from a top-down view in the centre, surrounded by location specific RGB camera images from the factory.

5.3 Baseline: Simulation-to-Real Domain Gap

Before applying any domain adaptation strategy, it is important to establish the gap between the synthetic training distribution and the real-world target domain. This section evaluates models trained purely on synthetic data, first on their own synthetic test splits to confirm in-domain competence, and then on the real-world test set to quantify the domain gap.

5.3.1 Synthetic Model: In-Domain Performance

A YOLO model trained on the synthetic dataset, comprising 21 camera viewpoints and 1000 frames per camera, achieved strong in-domain detection performance. This confirms that the synthetic data are learnable and that neither the YOLOv8m architecture nor the dataset size is the primary performance bottleneck.

Table 5.1: Synthetic model performance on synthetic test set (in-domain baseline).

Class	Precision	Recall	mAP50	mAP50-95
All	0.986	0.943	0.971	0.867
Forklift	0.989	0.949	0.977	0.907
Other vehicle	0.985	0.968	0.984	0.951
Pedestrian	0.983	0.912	0.953	0.744

5.3.2 Synthetic Model: Performance on Real Test Set

When evaluated on the real-world test set without any adaptation, model performance degrades substantially, as shown in Table 5.2. This establishes the sim-to-real domain gap that motivates the adaptation strategies explored in subsequent sections.

Table 5.2: Synthetic model evaluated on the real-world test set (domain gap baseline).

Class	Precision	Recall	mAP50	mAP50-95
All	0.348	0.151	0.227	0.093
Forklift	0.194	0.083	0.11	0.062
Other vehicle	0.671	0.276	0.465	0.169
Pedestrian	0.179	0.093	0.105	0.049

The model achieved an overall mAP50 of approximately 0.22 on the real test set, compared with 0.97 on the synthetic in-domain test set (Table 5.1). The real domain score is therefore about 23% of the in-domain score, indicating substantial performance degradation. This gap arises from the differences in texture, lighting, colors, camera intrinsics, and object appearance between the synthetic and real domains, and motivates the fine-tuning and style transfer experiments presented in Section 5.6. The pedestrian class exhibits the largest relative drop, consistent with its greater appearance variability and smaller size compared to other classes. The *other vehicle* class achieves the highest performance, suggesting a smaller domain gap than the other classes. A possible explanation is that, despite differences in texture and illumination between the synthetic and real datasets, vehicles in this class retain relatively consistent geometric characteristics, making them easier for the detector to recognize across domains. In contrast, forklifts exhibit greater appearance variability in the real dataset, including multiple forklift types and a wide range of object scales due to varying camera distances. These factors likely increase the domain gap and reduce generalization performance. Similarly, pedestrians remain challenging because of their high pose and appearance variability as well as smaller size.

5.4 Synthetic Dataset Design: Effect of Viewpoint Diversity

A key design decision when generating synthetic training data is the placement and orientation of cameras. This section investigates whether the diversity of viewpoints in the training set influences a model’s ability to generalize to unseen camera configurations. To isolate the effect of camera viewpoints, the experiments are conducted entirely within the synthetic domain while keeping all other factors constant, including the scene, object placement, rendering settings, and visual style. This controlled experimental design ensures that any observed differences in performance can be attributed primarily to viewpoint variation rather than other sources of domain shift. Although the objective is sim-to-real adaptation, directly evaluating viewpoint di-

versity on real-world data would make it difficult to determine whether performance changes arise from camera viewpoints or from other domain differences, such as textures, lighting, or sensor characteristics. By first studying viewpoint diversity under controlled synthetic conditions, the influence of camera placement can be isolated. The insights gained from these experiments can then inform the design of synthetic datasets intended for sim-to-real transfer.

Starting from the original dataset containing images from 21 camera viewpoints, three separate training sets were constructed, each containing seven cameras. Set A consists of viewpoints that closely resemble the camera configurations used in the real deployment environment, Set B contains substantially different viewpoints, and Set C includes a diverse selection of viewpoints covering a broader range of perspectives. Separate models are trained on each training set and evaluated on the test splits of all three sets to assess how viewpoint diversity affects cross-viewpoint generalization.

5.4.1 Single-Set Baselines

Table 5.3 reports the self-test performance of each individual 7-camera set, establishing the in-distribution ceiling for each configuration.

Table 5.3: Self-test baseline for each individual 7-camera synthetic set (train and test on same set).

Dataset	Precision	Recall	mAP50	mAP50-95
Set A (similar to real)	0.983	0.955	0.976	0.856
Set B (diff. cams set 1)	0.986	0.918	0.954	0.822
Set C (diff. cams set 2)	0.971	0.911	0.945	0.775

All three sets achieve high in-domain mAP50 (≥ 0.945), confirming that each dataset is individually learnable and that differences in cross-set generalisation cannot be attributed to differences in dataset quality or model capacity.

5.4.2 Cross-Set Evaluation

Table 5.4 presents the complete cross-evaluation matrix. Each row corresponds to the training set and each column to the test set. The diagonal entries represent the self-test performance previously reported in Table 5.3, while the off-diagonal entries measure generalization to unseen viewpoints.

Table 5.4: Cross-evaluation matrix: mAP50 for all train/test set combinations. Diagonal entries are self-test baselines.

Train ↓ / Test →	Set A	Set B	Set C
Set A	0.976	0.460	0.810
Set B	0.833	0.954	0.839
Set C	0.921	0.837	0.945

The results demonstrate that viewpoint selection has a substantial impact on cross-viewpoint generalization. The largest performance degradation occurs when the model trained on Set A is evaluated on Set B, where the mAP50 decreases from 0.976 on the self-test to 0.460, representing an absolute reduction of 0.516. In contrast, the model trained on Set C generalizes considerably better to Set B, achieving an mAP50 of 0.837 despite never observing those viewpoints during training.

These results suggest that the ability to generalize is influenced not only by the number of training viewpoints, which is identical for all three sets, but also by the diversity and coverage of those viewpoints. Training on a more diverse collection of camera perspectives appears to produce features that transfer more effectively to previously unseen viewpoints.

Table 5.5 provides per-class mAP50 values for the weakest cross-set transfer (A→B) and the strongest cross-set transfer (C→A).

Table 5.5: Per-class mAP50 for the weakest and strongest cross-set transfer conditions.

Condition	All	Forklift	Other vehicle	Pedestrian
A → B (weakest)	0.460	0.455	0.542	0.382
C → A (strongest)	0.921	0.946	0.949	0.868

The performance degradation observed for A→B is consistent across all three object classes. The pedestrian class experiences the largest decrease (mAP50 = 0.382), while forklifts and other vehicles also exhibit substantial reductions. Since all classes are affected, the results suggest that the performance loss primarily reflects a mismatch between the camera viewpoints represented in the training and test sets rather than a class-specific limitation.

5.4.3 Combined Camera Sets

Table 5.6 presents the performance of models trained on combined 14-camera datasets. Each model is evaluated both on its corresponding self-test set and on the remaining held-out camera set that was excluded during training.

Table 5.6: Cross-evaluation for combined 14-camera sets. Self-test results are also reported for completeness.

Train	Test	Precision	Recall	mAP50	mAP50-95
A+B (14 cams)	Self (A+B)	0.983	0.927	0.962	0.795
A+B (14 cams)	Set C	0.972	0.847	0.870	0.634
B+C (14 cams)	Self (B+C)	0.982	0.888	0.940	0.761
B+C (14 cams)	Set A	0.952	0.880	0.923	0.597

Increasing the number of training viewpoints generally improves or maintains cross-set performance. Training on the combined Sets A+B increases the mAP50 on Set C from 0.839 (training on Set B alone) to 0.870. Likewise, training on Sets B+C achieves an mAP50 of 0.923 on Set A, slightly improving upon the 0.921 obtained

when training on Set C alone. Although the improvement in the latter case is marginal, adding additional viewpoints does not reduce performance in either experiment.

The results also indicate that the benefit of adding cameras depends on the viewpoint coverage already present in the training data. Combining Sets A and B provides a noticeable improvement when generalizing to the unseen viewpoints of Set C, suggesting that the additional camera perspectives increase the diversity of the training distribution. In contrast, adding Set B to the already diverse Set C yields only a small improvement when evaluated on Set A, indicating diminishing returns once the training set already provides sufficient viewpoint coverage.

Taken together with the cross-set evaluation presented in Section 5.4.2, these experiments suggest that both the number and diversity of camera viewpoints influence generalization to unseen viewpoints. Increasing the number of cameras generally improves or maintains performance, provided that the additional viewpoints contribute new visual perspectives rather than redundant observations. For synthetic data generation, these findings indicate that expanding viewpoint coverage is likely to be more beneficial than concentrating cameras within a limited range of perspectives, making viewpoint diversity an important consideration when designing datasets for sim-to-real object detection.

5.5 Synthetic Dataset Design: Background and Appearance

This section examines background rendering and object appearance as potential contributors towards the sim-to-real gap. We then identify the important candidates that might be responsible for generalizing to real domain and then use these findings to inform dataset design. We also evaluate whether a Gaussian Splatting environment improves generalization to the real domain and whether it provides a suitable basis for the simulator.

5.5.1 Gaussian vs. Non-Gaussian Background Rendering

The synthetic background was generated using a Gaussian Splatting reconstruction of the real factory. To evaluate whether this choice of background representation affects detection performance, a controlled experiment was conducted in which two parallel datasets were generated with exactly the same camera placements, object positions, and scene configurations, differing only in background rendering environment: one dataset used the Gaussian Splatting environment, and the other used standard Unreal Engine rendering. The datasets are comprised of 1000 scenes and 9 different camera viewpoints. We also observed that background rendering has an effect on lighting. Although the lighting of the mesh environment could be controlled, the lighting encoded in the Gaussian Splatting representation could not be adjusted using the standard pipeline. Consequently, object appearance differed between the two environments even when object positions and placements were identical. This is a limitation of the current experimental design. Because of this limitation, the

experiment cannot isolate the effect of background rendering from the associated lighting differences. However, this experiment is valuable in determining how much an environment affects the detection performance while keeping all other factors constant and specifically if Gaussian Splatting environment is useful in reducing sim-to-real gap.



Environment with 3DGS



Mesh environment (non-Gaussian Splatting)

Figure 5.2: A side-by-side comparison between 3DGS and corresponding mesh environments

Table 5.7: Evaluation between Gaussian and Non-Gaussian background datasets.

Train	Test	Precision	Recall	mAP50	mAP50-95
<i>In-domain baselines</i>					
Gaussian	Self	0.989	0.984	0.991	0.915
Non-Gaussian	Self	0.994	0.974	0.989	0.914
<i>Cross-domain evaluation</i>					
Gaussian	Non-Gaussian	0.694	0.565	0.621	0.315
Non-Gaussian	Gaussian	0.757	0.434	0.527	0.249
<i>Real-domain evaluation</i>					
Gaussian	Real	0.283	0.133	0.209	0.0685
Non-Gaussian	Real	0.0368	0.0691	0.0252	0.00614

Both models perform near-equivalently on their own test split (≈ 0.99 mAP50), confirming that neither rendering style poses inherent detection difficulty in isolation. However, both models suffer severe performance degradation when evaluated across rendering styles. The Gaussian model tested on Non-Gaussian data drops from 0.991 to 0.621 mAP50, while the Non-Gaussian model tested on Gaussian data drops from 0.989 to 0.527. This suggests a distribution shift between different background rendering styles and it’s importance in dataset design.

On the real dataset, the model trained with the Non-Gaussian background achieved 0.0252 mAP50, whereas the model trained with the Gaussian Splatting background achieved 0.209 mAP50. This result suggests that Gaussian Splatting environment is useful in reducing sim-to-real gap and consequently suggests that the background rendering style transfers significantly better to real environment than mesh based

environment. This result supports the use of the Gaussian Splatting representation as the background-rendering environment rather than the mesh based environment in the evaluated pipeline.

These results demonstrate that background representation is a meaningful variable in synthetic dataset design: even when objects, camera positions, and scene layout are identical, a change in background rendering environment, together with the associated differences in illumination, resulted in a substantial change in performance. These results provide evidence that the choice of environment representation is associated with differences in sim-to-real transfer under the conditions investigated.

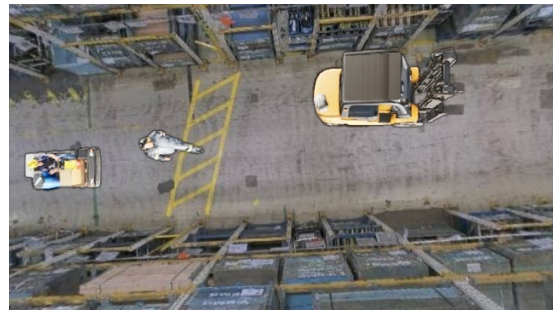
5.5.2 Real Background Augmentation on Synthetic Dataset

We observe that in our environment the background changes less frequently over time since it is an industrial environment with overhead cameras. Therefore, we can make use of real background environment if we have clean real images of background. In this case, we can make use of Cut-and-Paste augmentation method and leverage real background in our synthetic dataset while maintaining synthetic objects.

This section evaluates whether background cut-and-paste augmentation improves object detection and reduces the sim-to-real gap, and compares it with Gaussian Splatting rendering. We used the previously generated synthetic dataset and pipeline produced background masks to replace the synthetic backgrounds with real backgrounds. The resulting dataset retained the same camera and object configurations as the Gaussian Splatting dataset and differed primarily in background representation. It is worth noting that, because the real and synthetic camera poses are only approximate, discrepancies may occur between the background and synthetic objects which might slightly distort objects with respect to background or vice versa. Another limitation of this approach is the lighting discrepancy between Gaussian Splatting and augmented background since Gaussian Splatting representation produces it's own illumination which cannot be replicated with other backgrounds.



Reference real background without any objects



Real background augmented to synthetic dataset

Figure 5.3: A side-by-side comparison of real image and synthetic dataset augmented by it

Table 5.8: Evaluation of real augmentation vs. Gaussian Splatting

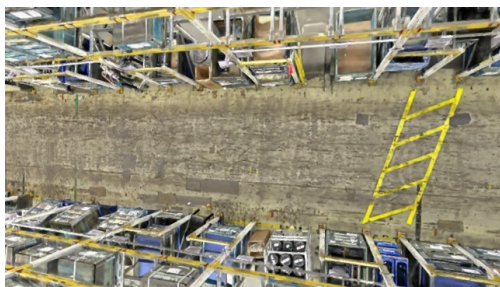
Train	Test	Precision	Recall	mAP50	mAP50-95
Background Cut-and-Paste	Real	0.247	0.291	0.199	0.0700
Gaussian Baseline	Real	0.283	0.133	0.209	0.0685

When evaluated on real images, Cut-and-Paste dataset achieves substantially higher recall than the Gaussian Splatting baseline. This suggests that training with real camera backgrounds may improve the model’s ability to detect objects in real images, although the underlying reason for this difference cannot be determined from this experiment alone. However, both approaches achieve comparable overall performance in terms of mAP, suggesting that Gaussian Splatting may provide a viable alternative to real background rendering for generating training data.

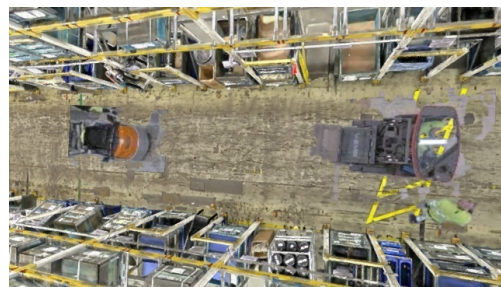
5.5.3 Object Appearances vs. Background Representation

The preceding experiments established that the Gaussian Splatting and conventional unreal engine rendering produce substantially different visual domains even when the simulated scene configuration is kept consistent. To further investigate object appearance, we use cut-and-paste augmentation: objects are extracted from the real dataset, stored in a per-camera object library, and pasted onto synthetic backgrounds. For this reason, the following twp datasets were generated with the same camera configuration and dataset size as before:

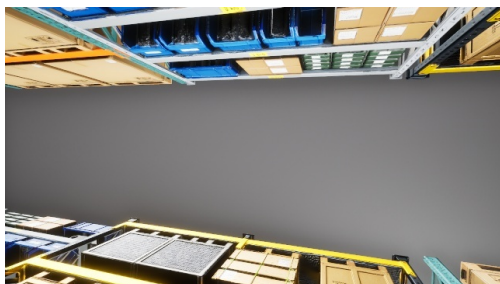
1. Gaussian Splatting environment with objects pasted from real cameras.
2. Unreal mesh environment with objects pasted from respective real cameras.



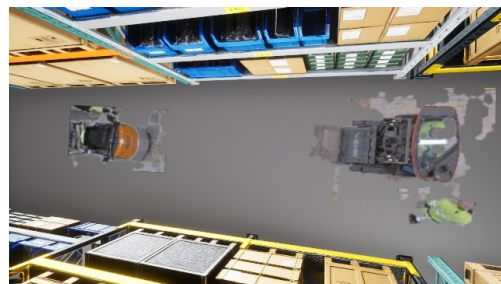
(a) Empty 3DGS background



(b) 3DGS with Cut-and-Paste



(c) Empty mesh background



(d) Mesh with Cut-and-Paste

Figure 5.4: Comparison of different background and object arrangements

Table 5.9: Models trained on four different combination of datasets evaluation metrics on real dataset

Model	Precision	Recall	mAP50	mAP50-95
GS background with C&P	0.713	0.138	0.291	0.158
Non-GS background with C&P	0.0755	0.0219	0.0473	0.0208
GS baseline	0.283	0.133	0.209	0.0685
Non-GS baseline	0.0368	0.0691	0.0252	0.00614

All four models were evaluated on the same manually annotated real-world test set consisting of 957 images from 11 ceiling-mounted cameras. The resulting mAP50 and mAP50-95 values are shown in Table 5.9. In the table, the "GS background with C&P" refers to the model trained on the Gaussian Splatting environment with real objects cut and pasted. Similarly, "Non-GS background with C&P" refers to the model trained on unreal mesh environment with real objects cut and pasted and other two are baseline models for Gaussian Splatting and unreal mesh with simulated objects.

From the results in Table 5.9, we observe that the representation of the background environment has a substantial effect on the detection performance. When synthetic objects were used, changing the background from conventional unreal rendering to Gaussian Splatting environment increases mAP50 performance from 0.0252 to 0.209 mAP50 which corresponds to the absolute improvement of 0.184 mAP while all other parameters were fixed. Similarly, when real objects were used through Cut-and-Paste augmentation, changing from the conventional Unreal Engine rendering to the Gaussian Splatting environment increased mAP50 from 0.0473 to 0.291, corresponding to an absolute improvement of 0.2437. This result highlights the importance of background representation and the potential of Gaussian Splatting to reduce the domain gap between simulation and real environment.

The results also provide insight into how object appearance affects detection performance. When unreal rendering environment was used, going from synthetic to real objects (Cut-and-Paste) increases performance from 0.0252 to 0.0473 mAP50, an absolute increase of 0.0221 mAP50. Similarly, when Gaussian Splatting rendering environment was used, going from synthetic to real objects increases performance from 0.209 to 0.291 mAP50 which is a 0.082 mAP50 performance increase. Although performance improved, the results indicate that object appearance alone, at least with the current cut-and-paste augmentation, is insufficient to close the domain gap since we only observed 0.0221 mAP50 increase in performance going from synthetic to real objects in unreal rendering environment. However, we observed a more substantial increase in Gaussian Splatting environment when we went from synthetic to real objects which indicate the possibility that the background representation and object appearance may interact, so their effects cannot be fully separated in the current experimental setup.

An important limitation of our Cut-and-Paste augmentation method is that it is not accurate in terms of geometric consistency and does have imperfections in the form of background artifacts and also happen to have color inconsistencies with respect to the target environment which could affect the detection performance. However, the

experiment remains useful as objects and backgrounds are evaluated under similar conditions and provide useful diagnosis for sim-to-real gap by varying objects and background rendering.

Overall, this experiment indicates that both object appearance and environmental background contribute to sim-to-real gap and it is important to consider these factors during dataset design. In particular, we found that Gaussian Splatting substantially improves real-world transfer compared with the conventional unreal background, even when the detected objects remain fully synthetic. This provides evidence that reconstructing the real factory environment using Gaussian Splatting is useful in synthetic dataset design for camera-based perception in industrial environment. We also observed that the background representation has the larger effect on performance under the conditions investigated and the object appearance and background may interact and affect detection performance. While the real objects combined with the Gaussian Splatting background further reduced the performance gap. This motivates future work on more realistic actor models.

5.6 Bridging the Sim-to-Real Gap

Having established the domain gap in Section 5.3, this section evaluates several approaches for reducing the overall gap, including domain adaptation methods and fine-tuning.

5.6.1 Fine-tuning on Real Data: Baseline

As the primary real data baseline, YOLOv8m pre-trained was fine-tuned on real factory dataset with five different random 60/20/20 splits on different random seeds. Because the real dataset was limited in size and performance estimates from small splits were noisy, Table 5.10 reports the mean performance across five random seeds, with standard deviation capturing split variance.

Table 5.10: Real-only fine-tuning: mean $\pm$ SD across five random seeds (seeds 8, 34, 42, 77, 85). Test set size varies slightly per seed due to rounding of the 60/20/20 split.

Class	Precision	Recall	mAP50	mAP50-95
All	0.816 ± 0.040	0.831 ± 0.036	0.876 ± 0.030	0.571 ± 0.039
Forklift	0.712 ± 0.062	0.897 ± 0.018	0.868 ± 0.044	0.586 ± 0.070
Other vehicle	0.916 ± 0.038	0.872 ± 0.036	0.931 ± 0.021	0.669 ± 0.026
Pedestrian	0.820 ± 0.071	0.724 ± 0.077	0.828 ± 0.034	0.458 ± 0.041

Fine-tuning on real images achieves a mean overall mAP50 of 0.876, a substantial improvement over the unadapted synthetic model (≈ 0.20) demonstrating that even a small real dataset is learnable. The variance across seeds is notable: the mAP50 range spans from 0.819 (seed 8) to 0.907 (seed 34), a difference of 0.088, reflecting the sensitivity of performance to how real images happen to be partitioned. Other vehicle achieves the highest mAP50 (0.931) and lowest variance (0.021), indicating

consistent and reliable detection. Pedestrian performance is lowest (0.828) and exhibits high precision variance (± 0.071), consistent with its status as the class with fewest instances and greatest appearance variation.

5.6.2 Synthetic Pre-training Followed by Real Fine-tuning

To evaluate whether synthetic pre-training provides a better starting point for real-data fine-tuning, YOLOv8m was first trained on the 21-camera synthetic dataset and subsequently fine-tuned on the same real data splits used in Section 5.6.1. Table 5.11 reports results under the same five-seed protocol.

Table 5.11: Synth→Real fine-tuning: mean $\pm$ SD across five random seeds. Model initialised from 21-camera synthetic pre-trained weights before fine-tuning on real data.

Class	Precision	Recall	mAP50	mAP50-95
All	0.850 ± 0.025	0.858 ± 0.027	0.891 ± 0.028	0.588 ± 0.023
Forklift	0.767 ± 0.095	0.885 ± 0.071	0.884 ± 0.035	0.632 ± 0.045
Other vehicle	0.905 ± 0.030	0.899 ± 0.043	0.928 ± 0.037	0.665 ± 0.026
Pedestrian	0.877 ± 0.036	0.789 ± 0.054	0.862 ± 0.035	0.468 ± 0.031

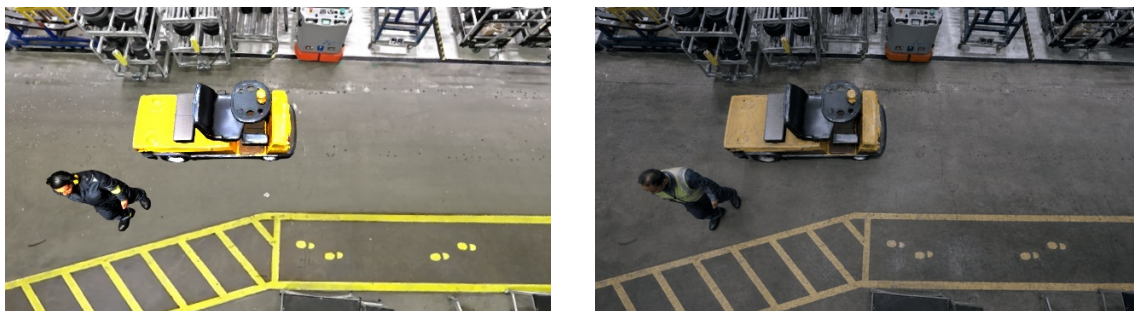
Table 5.12 directly compares the two fine-tuning conditions.

Table 5.12: Direct comparison of real-only fine-tuning vs. synth→real fine-tuning (mean mAP50 across five seeds). Δ = synth→real minus real-only.

Class	Real-only mAP50	Synth→Real mAP50	Δ
All	0.876 ± 0.030	0.891 ± 0.028	+0.015
Forklift	0.868 ± 0.044	0.884 ± 0.035	+0.016
Other vehicle	0.931 ± 0.021	0.928 ± 0.037	−0.003
Pedestrian	0.828 ± 0.034	0.862 ± 0.035	+0.034

Synthetic pre-training yields a consistent directional improvement of +0.015 mAP50 overall across all five seeds. However, this improvement is smaller than the within-condition standard deviation (± 0.030). The class-level results show a similar pattern, with no substantial improvement over real-only fine-tuning. This may indicate either that synthetic pre-training adds little useful information beyond the pre-trained YOLO backbone or that some synthetic-domain information is lost during two-stage fine-tuning. This experiment is insufficient to determine the cause. Under the tested conditions, the observed improvement was small relative to split-to-split variability, so the results do not provide clear evidence that synthetic pre-training outperforms standard YOLO fine-tuning.

5.6.3 Domain Adaptation via Style Transfer



Before style transfer

After style transfer

Figure 5.5: A side-by-side comparison between source and style transferred image.

Given the large domain gap observed in Section 5.3 and the limited improvement from two-stage fine-tuning, we employed a style transfer technique conditioned on a reference image and a text prompt as discussed in previous chapters.

It was applied to the synthetic dataset as a domain adaptation method to potentially reduce sim-to-real gap. A set of approximately 400 style-transferred images were generated using this method conditioned on style supervision by prompts and reference real images. Figure 5.5 shows an example of style transfer applied to a reference synthetic image.

The style-transfer process adapts the environmental style and colors as well as object appearance. Although the method was not always geometrically consistent and required manual supervision, it still produced a substantial improvement in real-domain performance. The limitations included slow processing as well as manual supervision. Consequently, only a relatively small style-transferred dataset could be generated and therefore a dataset consisting of only around 400 style transferred images split over 6 different camera configurations was generated. Although the number of cameras and the dataset size were limited relative to the real dataset, the experiment still provides useful evidence about whether diffusion based style transfer is a promising approach. For this reason, models were fine-tuned on the source synthetic dataset and the style-transferred dataset, and their performance was compared on the real test set. The images are derived from the synthetic dataset but are stylistically transformed to resemble the real factory’s appearance. Table 5.13 summarizes the impact of style transfer on real test set performance.

Table 5.13: Style transfer vs synthetic model evaluated on real test dataset.

Train	Test	Precision	Recall	mAP50	mAP50-95
Style transfer	Real	0.672	0.382	0.440	0.234
Synthetic baseline	Real	0.1770	0.0886	0.1250	0.0615

Style transfer alone increases performance on the real test set from approximately 0.125 to 0.44 mAP50, representing a gain of approximately 0.315 mAP50. Because

this dataset is much smaller, it is not directly comparable with the earlier datasets. Nevertheless, it achieved the highest zero-shot real-domain mAP50 among the conditions evaluated in this experiment, providing evidence that diffusion-based stylistic adaptation can reduce the domain gap when annotated real data are unavailable.

Table 5.14: Direct comparison of fine-tuning on synthetic dataset and equivalent style transferred dataset. Δ = mAP50 difference between synthetic vs. style transfer.

Class	Synthetic mAP50	Style Transfer mAP50	Δ
All	0.1250	0.4400	+0.3150
Forklift	0.0224	0.4680	+0.4456
Other vehicle	0.2600	0.5000	+0.2400
Pedestrian	0.0929	0.3510	+0.2581

Table 5.14 shows per-class mAP50 performance difference between style transfer dataset and source synthetic dataset. Performance increased for every class, with the forklift class showing the largest improvement. This suggests that style transfer is a promising approach for improving both overall mean average precision and per-class performance.

For completeness, Table 5.15 shows the metrics per class for the style transfer model evaluated on the real dataset and Table 5.16 shows same the metrics before style transfer.

Table 5.15: Performance of style transfer model on the real test set.

Class	Precision	Recall	mAP50	mAP50-95
all	0.672	0.382	0.440	0.234
forklift	0.733	0.427	0.468	0.282
other vehicle	0.632	0.437	0.500	0.281
pedestrian	0.650	0.283	0.351	0.139

Table 5.16: Performance of synthetic source model on the real test set.

Class	Precision	Recall	mAP50	mAP50-95
all	0.1770	0.0886	0.1250	0.0615
forklift	0.0175	0.0573	0.0224	0.0114
other vehicle	0.4280	0.0998	0.2600	0.1310
pedestrian	0.0856	0.1090	0.0929	0.0423

We next evaluate whether style transfer pre-training is beneficial when the resulting model is fine-tuned on the real dataset, analogous to the earlier synthetic pre-training experiment. However, due to limited real dataset, we split real dataset into two different splits for the ablation study:

1. Train: 30%, Validation: 10% and Test: 60%

2. Train: 15%, Validation: 5% and Test: 60%

Essentially, we keep test split constant and change the train and validation set for two ablation studies. For a fair comparison, we first fine-tuned real dataset on the standard yolo model and then fine-tune same splits on pre-trained style transferred model we trained earlier in Table 5.13.

Table 5.17: Comparison of the style-transfer model and baseline YOLO models fine-tuned and evaluated on the real dataset.

Model	Precision	Recall	mAP50	mAP50-95
<i>Splits: 30/10/60</i>				
Base yolo	0.825	0.678	0.774	0.434
Pre-trained style transfer	0.792	0.667	0.764	0.451
<i>Splits: 15/5/60</i>				
Base yolo	0.651	0.603	0.609	0.326
Pre-trained style transfer	0.678	0.631	0.647	0.360

We observe from Table 5.17 that average mAP50 performance on the first split is slightly lower for the pre-trained model than for the baseline YOLO model, while the mAP50-95 performance is slightly higher. For the second split, this difference is larger with pre-trained model achieving higher performance in both mAP50 and mAP50-95. However, the absolute difference is small, ranging from approximately 1 to 4 percentage points across the reported metrics. These results indicate that pre-training provided limited benefit in terms of average mAP, consistent with the earlier synthetic pre-training experiments. Based on these observations, pre-training appears to provide limited or no benefit for bridging the domain gap under the tested conditions even though in some cases it does improve performance slightly.

5.6.4 Joint Training by Data Mixing

In this section, we utilize the datasets we prepared earlier from section 5.6.3 to derive new datasets by mixing real, synthetic, and style transferred datasets and utilize them in joint training as opposed to two stage fine-tuning. In particular, we create the following new splits:

1. Style Transferred Dataset + Real (*Train: 30%, Val: 10%*)
2. Style Transferred Dataset + Real (*Train: 15%, Val: 5%*)
3. Synthetic Baseline Dataset + Real (*Train: 30%, Val: 10%*)
4. Synthetic Baseline Dataset + Real (*Train: 15%, Val: 5%*)

We train on the datasets above and test on same real test split i.e. 60% from section 5.6.3 and then also compare with two stage fine-tuning experiments.

Table 5.18: Comparison of training strategies on the real test set. The reported splits refer to the real dataset only; the synthetic and style-transfer datasets were kept constant across experiments.

Training Strategy	Precision	Recall	mAP50	mAP50-95
<i>Real split: 30/10/60</i>				
Real only	0.825	0.678	0.774	0.434
Sequential style transfer + real	0.792	0.667	0.764	0.451
Joint style transfer + real	0.874	0.805	0.864	0.653
Joint synthetic + real	0.850	0.766	0.843	0.579
<i>Real split: 15/5/60</i>				
Real only	0.651	0.603	0.609	0.326
Sequential style transfer + real	0.678	0.631	0.647	0.360
Joint style transfer + real	0.807	0.740	0.793	0.515
Joint synthetic + real	0.755	0.700	0.753	0.497

The results in Table 5.18 demonstrate that the effectiveness of synthetic data depends strongly on how it is incorporated into the training process. Sequential style-transfer pre-training followed by fine-tuning on the real dataset resulted in only marginal improvements compared with real-only fine-tuning. For the 30/10/60 split, mAP50-95 increased from 0.434 to 0.451, while mAP50 slightly decreased from 0.774 to 0.764. A similar but somewhat larger improvement was observed for the 15/5/60 split, where mAP50-95 increased from 0.326 to 0.360. These results suggest that using synthetic data solely as a preceding pre-training stage provides limited benefit when the model is subsequently fine-tuned using only the small real dataset.

In contrast, jointly training on synthetic and real data resulted in substantially higher performance. For the 30/10/60 split, joint training with style-transferred synthetic data increased mAP50-95 from 0.434 to 0.653, while also improving precision, recall, and mAP50. The same trend was observed for the 15/5/60 split, where mAP50-95 increased from 0.326 to 0.515. This indicates that synthetic data provides complementary training information that is not fully captured by the limited real dataset alone. Joint training allows the model to benefit from the additional variation provided by synthetic images while simultaneously using real images to maintain alignment with the target domain.

Furthermore, the style-transferred synthetic data consistently outperformed the original synthetic source data when combined with real images. This suggests that the visual similarity between the synthetic and real domains plays an important role in determining how effectively synthetic data can contribute to training. By reducing the visual domain gap, style-transferred images appear to provide a more useful complementary training signal for the target real-world domain.

An important finding is that joint synthetic–real training can also reduce the reliance on real annotated data. With the 15/5/60 split, which contains substantially less real training data, joint training with style-transferred synthetic data achieved an mAP50-95 of 0.515. This is higher than the 0.434 achieved by real-only fine-tuning using the larger 30/10/60 split. Although these results do not establish the minimum amount of real annotation required, they demonstrate that synthetic data can partially compensate for a reduction in real annotated training data while

maintaining or improving detection performance.

Overall, the experiments indicate that joint synthetic–real training is a more effective strategy than sequential synthetic pre-training for the evaluated object detection task. The results further suggest that carefully designed synthetic data can serve as a complementary source of training information and reduce the reliance on costly real-world annotations, particularly in low-data scenarios.

5.6.5 Pseudo-Labelling for Domain Adaptation

Pseudo-labelling is a semi-supervised learning technique that can be used in unsupervised domain adaptation (UDA) to exploit unlabelled target-domain data and help bridge the sim-to-real domain gap [31]. In this setting, a model trained on labelled source-domain data is used to generate predictions for unlabelled target-domain data. Predictions with sufficiently high confidence can then be treated as pseudo-labels and used for further training. This approach has also been applied in unsupervised domain adaptation to exploit unlabelled target-domain samples [32]. In this work, pseudo-labelling was performed after the style transfer stage. The style transfer model was selected as the initial model since it achieved the best performance prior to pseudo-labelling. This model was applied to the unlabelled real images, and detections with a confidence score greater than 0.7 were retained as pseudo-labels. Detections below this threshold were discarded to reduce the inclusion of potentially incorrect labels.

The resulting pseudo-labelled data were subsequently used to fine-tune the style transfer model. This allowed the model to learn from the characteristics of the real target domain without requiring manually annotated real images. The results of this process are presented in Table 5.19.

Table 5.19: Pseudo-labelling results: style transfer model applied to unlabelled real images; high-confidence detections retained as pseudo-labels for further fine-tuning.

Condition	P	R	mAP50	mAP50-95
Style transfer model (pre pseudo-label)	0.672	0.382	0.440	0.234
After pseudo-label fine-tuning	0.709	0.371	0.504	0.280

As shown in Table 5.19, the style transfer model achieved an mAP50 of 0.440 before pseudo-label fine-tuning. Following fine-tuning, mAP50 increased to 0.504, representing an improvement of 0.064. Similarly, mAP50-95 increased from 0.234 to 0.280. Precision also improved from 0.672 to 0.709, while recall decreased slightly from 0.382 to 0.371. Overall, the increase in both mAP50 and mAP50-95 indicates that pseudo-labelling provided a further improvement in detection performance on the real target domain. These results suggest that iterative pseudo-labelling may be worth investigating. However, in practice, many published studies use online pseudo-labeling, where the model is trained jointly on labeled synthetic data and unlabeled real-world data. As training progresses, the model’s performance and the

quality of the pseudo-labels increase; DACS: Domain adaptation via cross-domain mixed sampling [33] is one such example.

5.7 Summary of the Results

The experiments in this chapter provide evidence that the effectiveness of synthetic data for industrial object detection depends on how closely synthetic data represents the target domain and how effective it is for perception in real world. We initially found large sim-to-real gap and consequently investigated different dataset design factors such as background representation, objects appearance and effect of camera viewpoints. Furthermore, we investigated some domain adaptation techniques to reduce the domain gap and finally investigated two training strategies combining real and synthetic data and their effect on real world performance.

5.7.1 The Sim-to-Real Gap

The initial experiments establish that high performance on synthetic data doesn't necessarily translate to high performance on real factory images. This is known as simulation-to-real gap which gave opportunity to investigate different methods and factors to reduce this gap. The model trained on fully synthetic dataset consisting of 21 cameras and 1000 scenes achieved a mAP50 of 0.971 in it's own domain. However, same model performed poorly on real dataset achieving only around 0.23 mAP50 as shown in Figure 5.6.

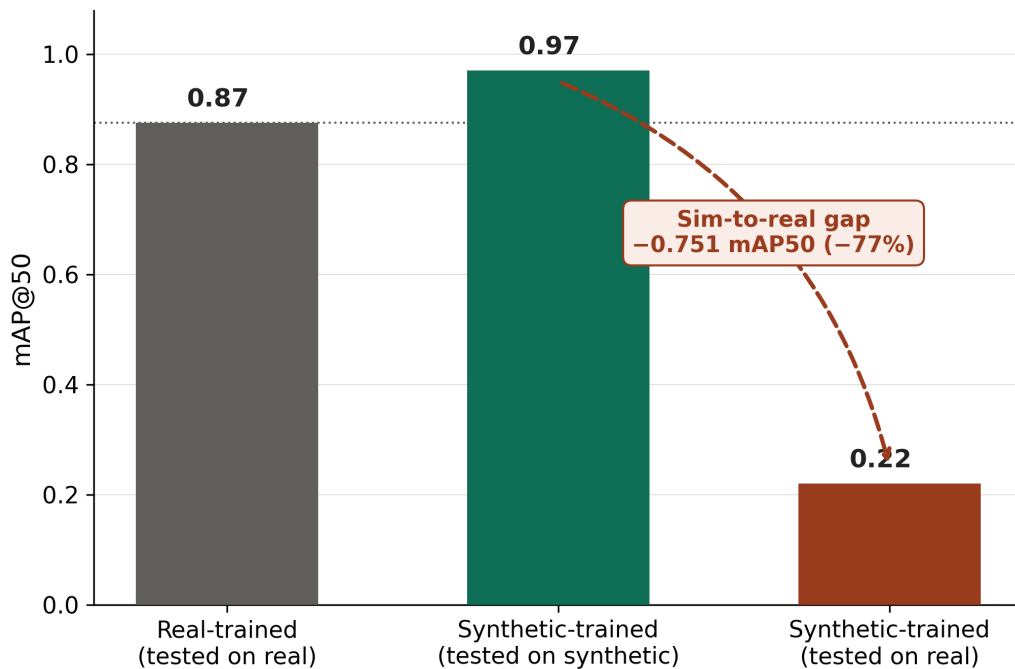


Figure 5.6: Visualizing sim-to-real domain gap.

This large difference demonstrates that the principal limitation is not the ability of the detector to learn from the synthetic images, but the mismatch between the

synthetic and the real target domain. Therefore, simply increasing dataset size or cameras was not enough to bridge the gap beyond a certain limit.

5.7.2 Importance of Synthetic Dataset Design Factors

We studied three main design factors for synthetic datasets which affected cross-domain performance between simulation and real environment. These three factors were camera viewpoints, background environment, and object appearance.

The results from controlled viewpoint experiments demonstrated that the camera configuration is an important component of synthetic dataset design. Although the three 7-camera datasets achieved similarly high in-domain performance, their ability to generalize to unseen viewpoints differed substantially. In the most pronounced case, a model trained on Set A achieved an mAP50 of 0.976 on its own test set but only 0.460 when evaluated on Set B. In contrast, the model trained on the more diverse Set C achieved 0.837 on Set B despite not being trained on those viewpoints. These results indicate that the distribution of camera viewpoints is important. Combining camera sets generally improved or maintained performance. This suggests that modeling cameras accurately to the real environment is important and, furthermore, a systematic study is needed to determine if diverse unseen viewpoints could be potentially beneficial in generalization.

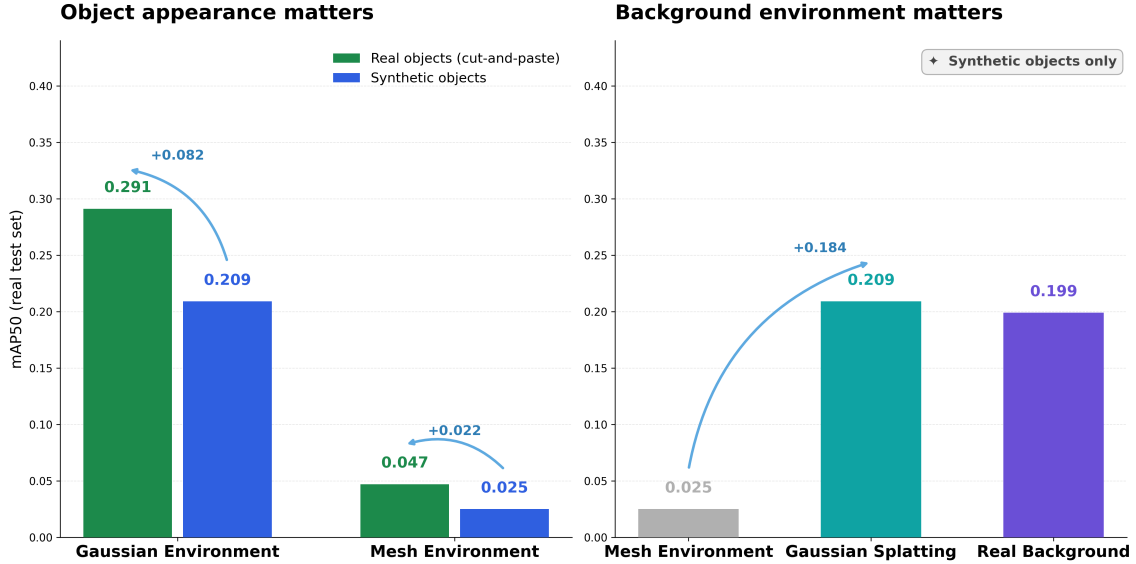


Figure 5.7: Figure showing background and object appearance results side-by-side.

The background experiments provide complementary evidence that the visual representation of the environment has a substantial effect on the transfer. We studied effect of gaussian and non-gaussian (simple mesh) background environments on sim-to-real transfer. The performance differed considerably when tested on real domain with the Gaussian Splatting model achieving substantially higher performance i.e. mAP50 of 0.209 as compared to 0.0252 of mAP50 for the conventional mesh-based

rendering environment while keeping synthetic objects and their positions as well as camera poses consistent across datasets. This also suggests a promise in using Gaussian Splatting as background representation as opposed to conventional simple mesh environment. Furthermore, the controlled object appearance experiments using cut and paste augmentations provide further insights. First, it suggests that object appearance also affects the sim-to-real performance, with real objects always performing better than synthetic objects regardless of the background environment. However, the results also indicate that background environment rendering and object appearance cannot be treated separately and they affect each other i.e. the gap between real and synthetic objects when using mesh background environment was 0.022 mAP50 while this gap was much bigger at 0.082 mAP when Gaussian Splatting environment was used. Therefore, while both object appearance and background environment matter, they cannot be treated separately but are rather complementary to each other as shown in Figure 5.7.

5.7.3 Domain Adaptation for Reducing Sim-to-Real Gap

We studied two domain adaptation methods. First, the style-transfer experiment which demonstrates that generative image-to-image translation is a promising approach in reducing sim-to-real gap. Particularly, we constructed a smaller dataset with fewer images and cameras due to limitations associated with our style transfer method. The baseline of this synthetic dataset achieved an mAP50 of 0.125 on the real test set, whereas the corresponding style-transferred dataset achieved 0.440. This represents an improvement of 0.315 mAP50 which represents best zero-shot performance improvement. Furthermore, the improvement was observed across all three object classes, with the largest absolute improvement occurring for forklifts. The style-transfer result is particularly important because it demonstrates that reducing the visual domain gap can substantially improve transfer.

The subsequent pseudo-labeling experiment suggests that the benefit of style transfer can be extended by exploiting unlabeled target-domain images. Fine-tuning using pseudo-labels increased mAP50 performance from 0.440 to 0.504 and mAP50-95 from 0.234 to 0.280. Precision also increased, while recall decreased slightly. Nevertheless, the modest decrease in recall and the reliance on a confidence threshold suggests that pseudo-labeling remains sensitive to the quality of the generated pseudo-labels.

The progression from the synthetic baseline through style transfer and pseudo-labeling is illustrated in Figure 5.8. The synthetic-only model provides a low baseline for real-world performance, while style transfer produces a substantial improvement without requiring manually annotated real training images. Pseudo-labeling provides a further improvement from 0.440 to 0.504 mAP50.

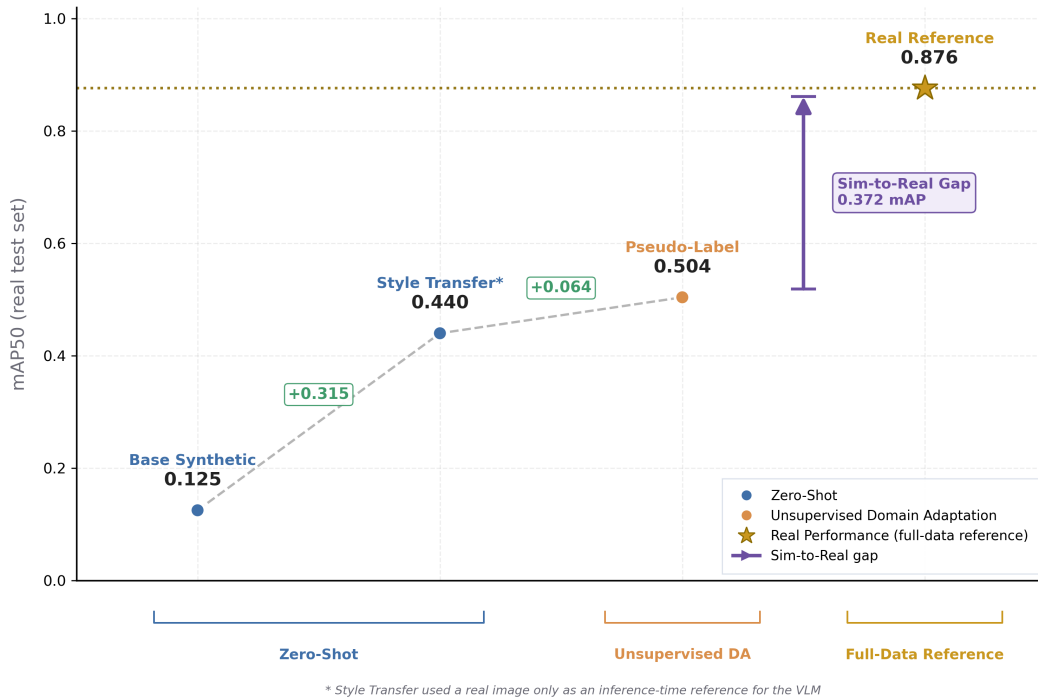


Figure 5.8: Progression of performance across different domain adaptation methods.

5.7.4 Annotation Efficiency

In experiments related to training strategy, we mainly study how we can incorporate a particularly small amount of real labeled dataset into training along with the synthetic dataset. The results show a clear distinction between sequential synthetic pre-training followed by real-data fine-tuning and joint synthetic + real training. For this study, we use two different real dataset train/val/test splits i.e. 30/10/60 and 15/5/60. Initializing the model from the style-transfer model and subsequently fine-tuning only on real images produced a lower mAP50 of 0.764. In contrast, joint training using the style-transferred synthetic data achieved 0.864. A similar pattern was observed for the 15/5/60 split: real-only training achieved 0.609, sequential training achieved 0.647, while joint style-transfer and real training achieved 0.793. Furthermore, using corresponding synthetic dataset before style transfer yielded similar pattern i.e. joint training performed substantially better than sequential fine-tuning.

The comparison between the two joint-training conditions provides further evidence that the quality of the synthetic data matters. At same real dataset splits, style transfer dataset performs slightly better overall than raw synthetic during joint training. This indicates that synthetic data is more useful when its visual characteristics are sufficiently aligned with the target domain.

The results can be interpreted as the evidence for improved annotation efficiency. When the amount of real annotated dataset is limited, within the experimental conditions evaluated, incorporating synthetic data improved sim-to-real performance. Or in other words, it reduced the amount of annotated real dataset required espe-

cially when joint training strategy is employed. This should not be interpreted as establishing a universal annotation requirement or a precise annotation reduction factor, but it provides direct evidence that synthetic data can partially compensate for limited real-world annotation given suitable training strategy. These results are summarized in Figure 5.9.

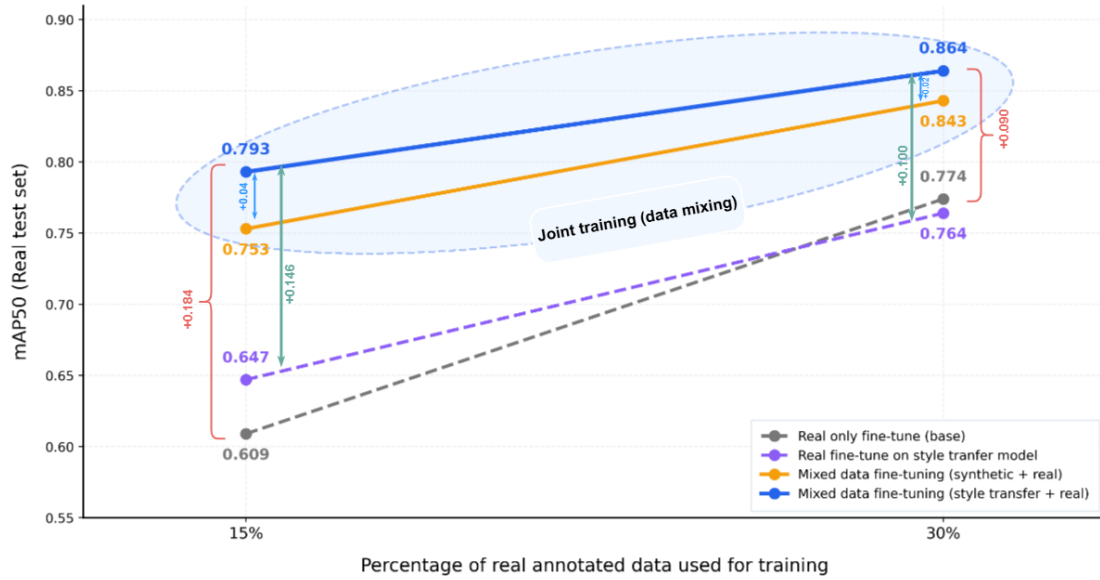


Figure 5.9: Effect of training strategy and availability of real annotated data on mAP50 evaluated on the real test set. Solid lines represent joint training with synthetic and real data, while dashed lines represent training using only real data or sequential synthetic pre-training followed by real fine-tuning.

6

Conclusion

This thesis investigated whether synthetic data can support camera-based perception in a large-scale industrial environment. In particular, we investigated whether a 3D Gaussian Splatting based digital twin is useful in synthetic data generation for ceiling cameras in industrial environment to support object detection and classification tasks. Furthermore, we studied the sim-to-real gap and some factors in dataset design and training which influence the usefulness of synthetic data. Finally, we investigated a few domain adaptation techniques which helped in reducing the domain gap between simulation and reality. Taken together, the results demonstrate that synthetic data can provide substantial value for industrial perception, but its effectiveness strongly depends on how well the synthetic data represents the target domain and how it is incorporated into the training process.

The initial experiments demonstrated that the synthetic data is learnable in its own domain but does not transfer well to the real domain and hence contributes towards a large sim-to-real gap. The synthetic environment was not inherently difficult for the detector but rather the principal challenge was transferring learned representations across domains. Therefore, we concluded that simply generating a large and correctly labeled synthetic data is insufficient due to visual discrepancy between the synthetic and real environments. We then investigated some factors in the dataset design which influenced the sim-to-real transfer. In particular, we studied camera viewpoints, object appearance and background rendering in controlled conditions.

The results demonstrated that the representation of the environment has a major influence on the sim-to-real transfer. When Gaussian Splatting and conventional unreal engine rendering were compared, the Gaussian Splatting representation provided substantially better transfer to the real environment under investigated conditions. This indicates that preserving the visual characteristics of the real environment through Gaussian Splatting reconstruction is beneficial and proved to be an important consideration in our synthetic dataset design. Further experiments also show that both the environment and the appearance of the objects contribute to the domain gap. Introducing real object appearances through Cut-and-Paste augmentation improved detection when used with the Gaussian Splatting environment. However, the improvement was much less pronounced when the same object treatment was combined with the conventional rendered background. This suggests that the object appearance cannot be considered independently from the surrounding environment i.e. improvements to object realism are more effective when the background and overall visual context are already reasonably aligned with the tar-

get domain under investigated conditions. The results therefore support treating the scene as a combination of interacting factors and point towards a future opportunity to improve object realism along with current Gaussian Splatting based environment.

We then explored some domain adaptation techniques since synthetic dataset design alone was found to be insufficient to close a large domain gap under investigated conditions. We particularly found out that bringing visual characteristics of synthetic images closer to real images can significantly improve their usefulness. Among the investigated approaches, diffusion based text to image generative models showed particularly promising results for improving zero-shot transfer to real images and consequently bridge a substantial sim-to-real gap. This is important because it demonstrates that synthetic data does not necessarily need to reproduce every visual property of the real environment perfectly in order to become useful. Reducing the visual discrepancies can already provide substantial benefits. At the same time, the limitations indicate that visual adaptation must be considered alongside computational cost, geometric consistency, and scalability.

We also performed experiments that provide an important insight into how synthetic data should be incorporated into training. Sequential synthetic pre-training followed by fine-tuning on real images provided only limited and inconsistent benefits compared with training on real data alone. In contrast, jointly training by mixing synthetic and real data produced considerably stronger improvements, particularly when the amount of labeled real data was limited. This points out that synthetic data can be used alongside a relatively small labeled real dataset to improve detection performance. In other words, the potential benefit of synthetic data is best understood in terms of annotation efficiency as it can help reducing dependency on real data and could reduce considerable time and effort in collection and annotation of real dataset.

Finally, we explored pseudo-labeling which is a semi-supervised learning technique that can be used in Unsupervised Domain Adaptation (UDA) to exploit unlabeled target-domain data. The pseudo-labeling experiment indicates that the synthetic-data pipeline can be extended beyond the use of labeled real images. After adapting the synthetic model towards the real domain, its predictions were used to obtain additional training information from unlabeled real images. The observed improvement suggests that this provides a promising direction for combining synthetic data generation, domain adaptation, and semi-supervised learning.

Overall, the findings of this thesis demonstrate that synthetic data generation based on 3D Gaussian Splatting reconstruction can be a useful component in camera-based perception for autonomous mobile robots in large-scale industrial environments. However, its value depends on dataset design, training strategy and domain alignment. This work highlights various factors such as environment representation, object appearance, viewpoint coverage, visual adaptation, and the strategy used to combine synthetic and real data all influence the final performance. Finally, the results not only provide the evidence for the usefulness of a 3D Gaussian Splatting-

based digital twin in synthetic data generation, but also the practical insights into how such a synthetic environment can be utilized more effectively for perception in a real industrial environment.

7

Limitations, Ethics and Future Work

We identified several limitations in this work which provide directions for the future work. First, the camera configurations should be modeled more accurately. The synthetic cameras used in this study provide only an approximate representation of the real camera setup in the factory, and substantial discrepancies exist between some synthetic and real camera configurations. In particular, the camera positions, orientations, and number of cameras should be modeled to more closely reflect the actual deployment scenario. This is particularly important because the viewpoint experiments demonstrate that camera configuration can significantly affect detection performance, even when the surrounding environment remains unchanged. Future work should therefore reproduce the real camera configurations as accurately as possible and subsequently investigate controlled perturbations around the nominal camera poses. Such experiments could provide a systematic assessment of the extent of viewpoint variation that can be tolerated while maintaining robust detection performance in a real-world deployment.

Second, due to technical limitations in Unreal Engine and the Gaussian Splatting, it was not possible to perform conventional lighting related domain randomization since Gaussian Splatting contains its own illumination and does not react to external illumination in our specific setup and would therefore need modifications in technical implementation. Consequently, variations in background representation also affect the illumination and appearance of the inserted objects. This limitation also makes it difficult to isolate background geometry, texture, and illumination as completely independent factors. A future Gaussian Splatting pipeline should investigate methods to make Gaussian Splatting react to external illumination inside the engine and, therefore, allow controlled variations in lighting, shadows, colors and other domain randomization techniques allowing their individual contributions to sim-to-real transfer to be systematically evaluated. Previous research has demonstrated that randomizing visual properties such as lighting and textures can improve the robustness of models trained in simulation and facilitate transfer to real-world environments [34, 35].

Third, the current pipeline uses Gaussian Splatting primarily to represent the static background environment while the objects are represented using synthetic assets. As we identified from experiments related to object appearance, there is a potential for improvement in object appearances. An interesting extension would be to

investigate Gaussian Splatting representations for objects themselves. Object-level Gaussian representations could potentially provide more realistic textures and appearance characteristics than the generative 3D assets used in the current pipeline and might potentially help in reducing sim-to-real domain gap.

Generative style-transfer results motivate further investigation into scalable and geometrically constrained image transfer. The improvement demonstrates considerable potential, but the current approach required manual supervision and therefore resulted in only a relatively small dataset because image generation was slow and was not always geometrically consistent. A future approach should explore geometrically consistent methods and study how the amount of style-transferred dataset affects the domain gap. One promising approach is to incorporate a recent image generation model from Alibaba known as Qwen-Image-2.1 [36]. It is particularly interesting since it’s an open weight model and brings image generation, transparency, and a wide range of editing capabilities together in one model. Faster inference, support for up to 10 reference images, flexible local editing, and several other features such as using masks for editing, which is a natural extension of our current pipeline since our pipeline includes several complementary components.

Furthermore, other methods for domain adaptation such as Domain Adversarial Neural Networks (DANN) [37], Correlation Alignment (CORAL) [38] and better pseudo-labeling such as different confidence thresholds, class-specific thresholds, iterative pseudo-labeling, and methods for reducing confirmation bias could be explored. Imperfections in cut-and-paste experiments were also a limitation and better image blending techniques could be explored for cleaner experiments.

7.1 Ethics

Camera systems in factory environments introduce several ethical considerations. In this project, the camera system is primarily used for safety by enabling autonomous mobile robots to detect pedestrians, vehicles, and other obstacles and respond accordingly. This provides a strong ethical justification for the use of such systems, since insufficient perception could increase the risk of injury to pedestrians or damage to equipment. However, the same camera system can also function as a form of workplace surveillance. Since the cameras continuously observe the factory environment, they may capture information about employees and other pedestrians who have not explicitly consented to being recorded. Several measures can be used to reduce these privacy concerns. Access to recorded data can be restricted and the data can be protected against unauthorized use. More extensive measures, such as anonymization, blurring of individuals, or low-resolution processing, could further reduce privacy risks. However, such approaches may also remove visual information that is useful for object detection and could therefore reduce perception performance. A related ethical concern is the use of real factory images for training perception models. Including identifiable individuals in datasets without appropriate consent or safeguards raises privacy concerns. Synthetic data generated in a simulation envi-

ronment can reduce the need for real-world image collection and therefore reduce the amount of personal data included in training datasets. However, the results of this thesis indicate that synthetic data cannot fully replace real data, meaning that some real factory images are still required. A mixed dataset containing both synthetic and real images can nevertheless reduce the overall dependence on real-world data. This creates a trade-off between privacy and system safety. Since models trained on synthetic data still show lower real-world performance than models using real data, excessive reliance on synthetic data may reduce system reliability. Therefore, the use of synthetic data in safety-critical factory perception systems remains an ethical consideration as long as a substantial performance gap between synthetic and real data persists.

7.2 Conclusion

In conclusion, the results of this thesis indicate that Gaussian Splatting based synthetic data generation is a promising approach for industrial object detection, but its value depends on how synthetic domain is constructed and adapted. Particularly, we have three complementary findings: reproducing the real environment with Gaussian Splatting can substantially improve sim-to-real transfer, visual adaptation through style transfer can further reduce the domain gap, and jointly combining adapted synthetic data with a limited amount of real data. Future improvements in camera calibration, controllable lighting, domain randomization, object-level Gaussian representations, and scalable style transfer could therefore make the proposed approach substantially more robust and practical for real industrial deployment.

Bibliography

- [1] Jonathan Edvinsson Björkman, Sigrid Ekström, Filip Håkansson, Valerija Kazlauskaite, Adam Sunnar, and Hugo Wallin. Fotorealistic simulering av mobila robotar i en fabriksmiljö. Kandidatarbete EENX16-26-16, Chalmers tekniska högskola, Institutionen för Elektroteknik, Göteborg, Sverige, 2026.
- [2] Sascha Kirch. Turn yourself into a 3D gaussian splat: A hands-on guide for practitioners, March 2024. <https://towardsdatascience.com/turn-yourself-into-a-3d-gaussian-splat-3a2bc59a770f/>. Accessed: Jun. 19, 2026.
- [3] Ultralytics. Data augmentation using ultralytics YOLO. <https://docs.ultralytics.com/guides/yolo-data-augmentation/>. Accessed: Oct. 2, 2026.
- [4] RangeKing. Brief summary of YOLOv8 model structure. GitHub issue #189, Ultralytics/Ultralytics, January 2023. <https://github.com/ultralytics/ultralytics/issues/189>. Accessed: Oct. 2, 2026.
- [5] Volvo Group. AI transporters push the boundaries of modern manufacturing, 2024. <https://www.volvogroup.com/en/news-and-media/news/2024/nov/ai-modern-manufacturing.html>. Accessed: Jun. 19, 2026.
- [6] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.
- [7] Koji Aoshima and Martin Servin. Examining the simulation-to-reality gap of a wheel loader digging in deformable terrain. *Multibody System Dynamics*, 64:121–148, 2025.
- [8] Elie Aljalbout, Jiaxu Xing, Angel Romero, Ireteayo Akinola, Caelan Reed Garrett, Eric Heiden, Abhishek Gupta, Tucker Hermans, Yashraj Narang, Dieter Fox, Davide Scaramuzza, and Fabio Ramos. The reality gap in robotics: Challenges, solutions, and best practices. *Annual Review of Control, Robotics, and Autonomous Systems*, 9:403–432, 2026.
- [9] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):139:1–139:14, 2023.
- [10] Epic Games. Balancing blueprint and C++. Unreal Engine 4.27 Documentation. https://dev.epicgames.com/documentation/unreal-engine/balancing-blueprint-and-cplusplus?application_version=4.27. Accessed: Jun. 22, 2026.
- [11] Hussein Ahmed, Linnea Bodin, Emil Jansson, Lucas Nilsson, Melker Sjöstedt, and David Hafstad Wallander. Realistic träningsdata med 3D gaussian

- splatting: Bygga digitala tvillingar för generering av ground-truth-data i unreal engine 5. Bachelor's thesis, Chalmers University of Technology, 2025.
- [12] Mohammad Albarham, Jinghao Li, Dineth Gurusinghe, Anirudh Vinod Nair, Yahya Khalid, and Youli Gong. Gaussian splatting-based simulation for multi-view and BEV semantic learning. Technical report, Chalmers University of Technology, 2025.
 - [13] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio 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 (CVPR)*, pages 3234–3243, 2016.
 - [14] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 102–118, 2016.
 - [15] Aneesh Deogan, Wout Beks, Peter Teurlings, Koen de Vos, Mark van den Brand, and René van de Molengraft. Synthetic dataset generation for autonomous mobile robots using 3D gaussian splatting for vision training. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pages 2802–2807. IEEE, 2025.
 - [16] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016.
 - [17] Zoubin Bi, Yixin Zeng, Chong Zeng, Fan Pei, Xiang Feng, Kun Zhou, and Hongzhi Wu. GS³: Efficient relighting with triple gaussian splatting. In *SIGGRAPH Asia 2024 Conference Papers*, pages 12:1–12:12. Association for Computing Machinery, 2024.
 - [18] David Bernecker. Image processing. In Andreas Maier, Stefan Steidl, Vincent Christlein, and Joachim Hornegger, editors, *Medical Imaging Systems: An Introductory Guide*, volume 11111 of *Lecture Notes in Computer Science*, pages 37–55. Springer, Cham, Switzerland, 2018.
 - [19] Epic Games. Unreal engine. <https://www.unrealengine.com>. Accessed: Dec. 15, 2025.
 - [20] XGRIDS. XGRIDS Developer Documentation. <https://developer.xgrids.com/#/document?titleId=en-1720509312452>. Accessed: Jun. 22, 2026.
 - [21] OpenAI. Video generation models as world simulators. <https://openai.com/index/video-generation-models-as-world-simulators/>, February 2024. Accessed: Oct. 2, 2026.
 - [22] Adobe. Mixamo. <https://www.mixamo.com/>. Accessed: Sep. 2, 2026.
 - [23] Blender Foundation. Blender. <https://www.blender.org/>. Accessed: Sep. 2, 2026.
 - [24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
 - [25] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

-
- [26] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.
 - [27] Ultralytics. Ultralytics YOLOv8, 2026. <https://docs.ultralytics.com/models/yolov8/>. Accessed: Oct. 2, 2026.
 - [28] Ultralytics. COCO detection dataset, 2026. <https://docs.ultralytics.com/datasets/detect/coco/>. Accessed: Oct. 2, 2026.
 - [29] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, August 2026. Version 8.4.115, Zenodo. <https://doi.org/10.5281/zenodo.21742658>.
 - [30] Yasser Benigmim, Subhankar Roy, Slim Essid, Vicky Kalogeiton, and Stéphane Lathuilière. One-shot unsupervised domain adaptation with personalized diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 698–708, June 2023.
 - [31] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *ICML 2013 Workshop on Challenges in Representation Learning*, 2013.
 - [32] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6028–6039. PMLR, 2020.
 - [33] Wilhelm Tranheden, Viktor Olsson, Juliano Pinto, and Lennart Svensson. DACS: Domain adaptation via cross-domain mixed sampling. *arXiv preprint arXiv:2007.08702*, 2020.
 - [34] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.
 - [35] Fabio Muratore, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger, and Jan Peters. Robot learning from randomized simulations: A review. *Frontiers in Robotics and AI*, 9, 2022.
 - [36] Qwen Team. Qwen-image-2.1: Compact, efficient, and unified image creation, September 2026. <https://qwen.ai/blog?id=qwen-image-2.1>. Accessed: Oct. 2, 2026.
 - [37] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.
 - [38] Baochen Sun and Kate Saenko. Deep CORAL: Correlation alignment for deep domain adaptation. In *Computer Vision – ECCV 2016 Workshops*, pages 443–450. Springer, 2016.

A

Appendix 1

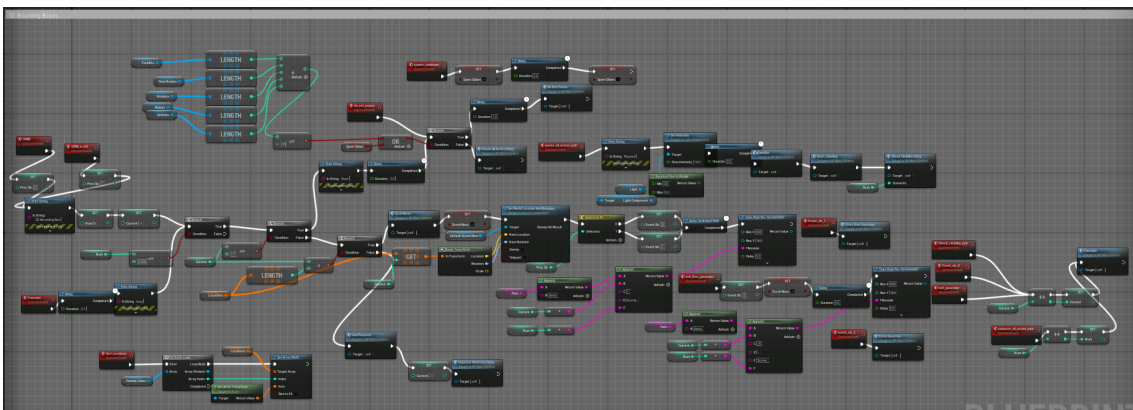


Figure A.1: Unreal Engine blueprint implementation for synthetic data collection.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY