Domain Randomization for Object Detection in Manufacturing Applications using Synthetic Data: A Comprehensive Study*

Xiaomeng Zhu[1,3], Jacob Henningsson[1], Duruo Li[1], Pär Mårtensson[1], Lars Hanson[2], Mårten Björkman[3], Atsuto Maki[3]

***Abstract*— This paper addresses key aspects of domain randomization in generating synthetic data for manufacturing object detection applications. To this end, we present a comprehensive data generation pipeline that reflects different factors: object characteristics, background, illumination, camera settings, and post-processing. We also introduce the Synthetic Industrial Parts Object Detection dataset (SIP15-OD) consisting of 15 objects from three industrial use cases under varying environments as a test bed for the study, while also employing an industrial dataset publicly available for robotic applications. In our experiments, we present more abundant results and insights into the feasibility as well as challenges of sim-to-real object detection. In particular, we identified material properties, rendering methods, post-processing, and distractors as important factors. Our method, leveraging these, achieves top performance on the public dataset with Yolov8 models trained exclusively on synthetic data; mAP@50 scores of 96.4% for the robotics dataset, and 94.1%, 99.5%, and 95.3% across three of the SIP15-OD use cases, respectively. The results showcase the effectiveness of the proposed domain randomization, potentially covering the distribution close to real data for the applications.**

I. INTRODUCTION

In advanced manufacturing, object detection plays a critical role in boosting operational efficiency and ensuring product quality. Tasks such as quality inspection and robotic picking rely on accurate object detection systems [1]. However, deploying deep learning models in manufacturing is challenging due to the limited availability of annotated real-world data, which is costly and time-consuming to gather.

An effective solution is to generate synthetic data from Computer-Aided Design (CAD) models, which are commonly used in manufacturing [2]. This method produces diverse datasets that replicate complex scenarios without the need for costly real-world data collection and annotation. Synthetic data reduces time, eliminates annotation errors, and enables the creation of rare or hard-to-capture events. However, the sim-to-real reality gap remains a major challenge. Models trained on synthetic data often struggle to generalize to real-world data due to differences in data distribution and environmental conditions [3].

One approach to bridging the reality gap is domain randomization (DR), which introduces random variations into synthetic data, training models to view real-world data as just another variant to enhance their generalization to real-world applications [4]. Initially proposed by Tobin et al. [4], DR has been adopted in several manufacturing applications [1], [2], [5], [6], [7], [8]. However, current methods face two limitations: a) exploring across limited object appearances and b) being evaluated on real data in controlled environments with consistent backgrounds. Manufacturing objects often have unique appearances and material properties, such as metallic surfaces or textureless finishes, that differ from everyday objects. Additionally, while manufacturing environments are generally more controlled than outdoor settings, they can still present varying scenarios, backgrounds, occlusions, and lighting inconsistencies. These factors can impact object detection performance, and we argue that current state-of-the-art methods have yet to fully capture the complexity of real manufacturing settings.

This study aims to identify key DR factors and develop a comprehensive synthetic data generation pipeline for manufacturing object detection applications. Previous literature shows that effective domain randomization (DR) typically applies to components such as object characteristics, background, illumination, camera settings, and post-processing. Building on this, we developed a pipeline that incorporates DR across all these dimensions and generates RGB images from CAD models. To evaluate its effectiveness, the pipeline was tested on two datasets representing complex manufacturing object appearances and scenarios. The first dataset, from Horvath et al. [8], is tailored for a robotic application featuring 10 industrial objects. The second is the Industrial Parts Object Detection Dataset (SIP15-OD) developed by us, which includes 15 objects from three distinct manufacturing use cases captured under varied environmental conditions.

To our knowledge, this is the first study to account for all DR components in synthetic data generation and evaluate them across multiple datasets for manufacturing object detection. This comprehensive study revealed several critical factors, such as object material properties and rendering methods, that are crucial for sim-to-real transfer in manufacturing object detection but have been underexplored. Leveraging these insights, we trained Yolov8 models only on generated synthetic data and achieved mAP@50 scores of 96.4%, 94.1%, 99.5%, and 95.3% on the robotics dataset and the three SIP15-OD use cases, respectively.

The contributions of our work can be summarized as follows:

1) We identified key DR factors crucial for sim-to-real

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Scania CV AB, Södertälje, Sweden. {xiaomeng.zhu, jacob.henningsson, duruo.li, par.martensson}@scania.com

[2]Skövde University, Skövde, Sweden. lars.hanson@his.se

[3]KTH Royal Institute of Technology, Stockholm, Sweden. {xiazhu, atsuto, celle}@kth.se

manufacturing object detection, including rendering methods, object material properties, post-processing, and distractors.
2) We presented the SIP15-OD dataset, comprising 321 images and 877 annotated objects across 15 industrial categories from three distinct manufacturing use cases. This dataset captures some complexity of real-world manufacturing settings and can serve as a useful benchmark for object detection. The dataset and code are publicly available.[1].
3) We developed a synthetic data generation pipeline that enhanced sim-to-real manufacturing object detection, with models trained solely on its synthetic data achieving top performance on a public industrial dataset. Unlike previous studies limited to single settings, we evaluated it across multiple manufacturing scenarios, offering richer experimental results and broader insights for DR research.

II. Related work

A. Domain adaptation and domain randomization

Domain adaptation (DA) and domain randomization (DR) are two strategies to address the reality gap in machine learning. DA bridges the gap between two domains by aligning their distributions through techniques such as feature space mapping or adversarial training[3], [4]. Within the context of synthetic data generation, it is described as creating photo-realistic images mimicking real conditions, which often requires high-fidelity simulations and substantial computational resources [4], [8].

DR, on the other hand, introduces random variations into synthetic training data, enabling models to generalize across diverse scenarios, including unseen ones. By randomizing generation settings, this approach reduces dependence on high-fidelity simulations [2], [4], [8]. Structured domain randomization (SDR), introduced by Prakash et al. [9], enhances DR by incorporating realistic scene layouts. It places objects in contextually accurate environments, such as cars on roads, allowing models to learn their spatial relationships.

B. Object detection

Recent advances in deep learning have led to significant improvements in object detection models, with EfficientDet, Yolov8, and DETR standing out for their strong performance in terms of both accuracy and speed [10], [11], [12]. Among these, Yolov8 [11] stands out for its high performance and lightweight design, making it suitable for manufacturing applications. For comprehensive overviews of recent developments in object detection, see [13], [14], and [15].

C. Domain randomization in manufacturing applications

Different studies have explored DR for synthetic data generation in manufacturing object detection. Tobin et al. [4] pioneered DR in robotic applications, using simple geometric objects like cylinders and squares simulated in MuJoCo [16] with VGG-16 [17] for object localization. They randomized factors like object position, distractors, colors, camera view, number of lights, and post-processing noise. They rendered synthetic images using rasterization via OpenGL [18], which efficiently converts 3D geometric shapes into 2D images by projecting them onto a view plane. However, their use of basic shapes and non-realistic textures with low-quality renderers limited the generalizability of their approach to more complex scenarios.

Eversberg and Lambrecht [2] generated synthetic images for turbine blade quality inspection in a shop floor environment using Faster-RCNN [19]. They randomized object poses, textures, backgrounds, camera settings, and lighting conditions in Blender [20], leveraging physics-based rendering (PBR) with path tracing for highly realistic images. Path tracing simulates light interactions by tracing rays, capturing shadows, reflections, and global illumination [21]. Their study highlighted the advantages of combining DR with SDR for manufacturing applications, finding that DR was effective for non-object elements like backgrounds and distractors, while SDR with realistic lighting and PBR textures enhanced object-related features. However, the study focused on single-object scenarios, overlooking the complexities of multi-object detection and lacking gravity simulations and post-processing techniques.

Horvath et al. [8] applied DR to a robotic application involving 10 industrial parts, with their 3D models and real data publicly available. They used PyBullet [22] for gravity simulation, rasterization rendering, and Yolov4 [23] for detection. Their DR setup involved randomizing object pose, texture, background, camera pose, and post-processing. They highlighted the significance of object texture, pose, and post-processing for effective model performance. However, illumination randomization was not included in their study.

In addition, Sampaio et al. [7] generated synthetic data for inspecting two motorbike parts, emphasizing the role of distractors and image resolution. Cohen et al. [5] and Tang et al. [6] focused on post-processing data augmentation rather than domain randomization. Cohen, targeting seven chair parts, highlighted the effect of blur, while Tang, detecting ten engine parts, emphasized object positioning and clutter. All studies conducted ablation experiments, showing that object detection performance improves with more synthetic data, but only up to a certain point, beyond which additional images offer no further accuracy gains.

III. Method

A. Domain randomization

Considering the key factors mentioned in previous work, DR techniques can be summarized in five components: object randomization (quantity, pose, texture), background and distractor randomization, camera randomization (various views), illumination randomization (environmental lights), and post-processing (noise, blur). Our synthetic data generation pipeline applies DR across all these areas, and to our knowledge, we are the first to fully integrate all of them. The most relevant DR parameters in our pipeline are:

[1]Code: https://anonymous.4open.science/r/SynMfg_Code-6FF5/README.md

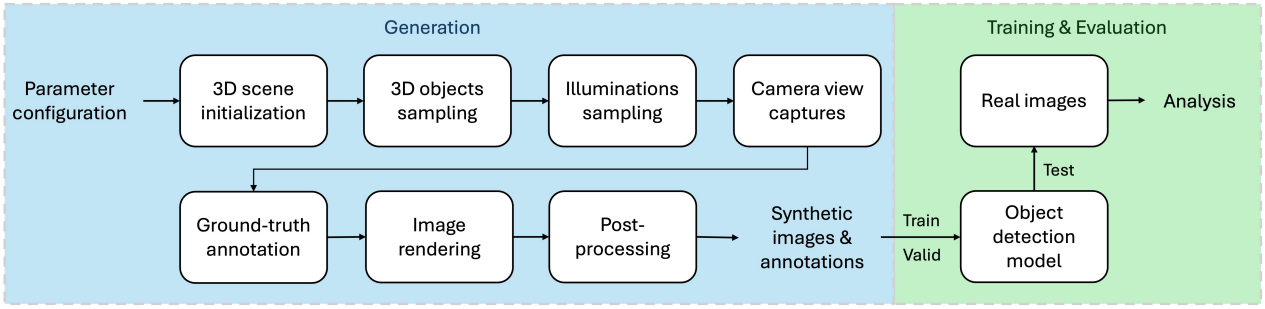


Fig. 1. Overview of our data generation pipeline and evaluation method. Processes shown in dark blue indicate where domain randomization is applied.

3D object sampling: ***Quantity:*** objects are randomly selected from a uniform distribution between zero and the total number of objects, with categories chosen evenly across available categories. ***Pose:*** Objects are assigned random coordinates and rotation, ensuring no overlap. ***Gravity point:*** The lowest points of objects are set to zero to simulate gravity, allowing various orientations to better represent real-world setups where objects do not always lie flat. ***Texture:*** Objects are assigned one of three textures: 1) RGB Texture: Random RGB values. 2) Image Texture: Random images from the Flickr 8K dataset [24]. 3) PBR Material Texture: Predefined materials from the CG Texture dataset [25], including properties such as metalness and roughness.

Camera view setup: ***Pose:*** Camera is positioned in spherical coordinates (radius r, polar angle θ, and azimuthal angle ϕ), with the radius determined by the camera sensor and the total object area $ObjectsArea$, which represents the bounding area covering all objects. These coordinates are then transformed into Cartesian coordinates (x,y,z) following

$$(x,y,z) = (r\cos\phi\sin\theta, r\sin\phi\sin\theta, r\cos\theta) \quad (1)$$

By uniformly sampling r, θ, and ϕ within specified ranges, the camera is randomly positioned on a spherical shell to control the object scale. ***Focus point:*** Camera aimes at the center of $ObjectsArea$ with random shifts along the XYZ axes. ***FOV:*** Focal length is randomized to control zoom.

3D scene initialization: ***Background:***

A 3D scene hosts the objects and camera, consisting of a ground plane and four vertical walls connected to its edges. The z-coordinate of the ground plane is set to zero, ensuring all objects are attached to it, simulating point gravity. The scene size is determined by the r of the camera and the $ObjectsArea$ to provide sufficient space. For background randomization, a randomly selected image from the BG-20K dataset [26] is applied to the ground plane and walls. ***Distractors:*** Six base 3D meshes (cubes, spheres, cones, etc.) from Blender [20] are added as other objects without gravity constraints as distractors.

Illuminations sampling: ***Quantity:*** Area lights are used, with the number determined by the scene size to ensure balanced lighting without over- or under-illumination. ***Energy & Color:*** Light intensity is randomly assigned, and the color is randomly selected from RGB values.

Post-processing: ***Noise & Blur:*** Random salt & pepper noise and Gaussian blur are applied with varying strength and probability.

B. Implementation details

Blender and its Python API (bpy) [27] are used to implement the domain randomization components. Fig. 1 shows the flow of our generation pipeline and evaluation method. Random settings are specified in a configuration file, and once selected, the pipeline generates synthetic data accordingly. During generation, segmentation masks and bounding boxes are created based on 2D pixel coordinates in the camera viewport, with masks saved as images and bounding boxes in YOLO format. For rendering, we use either the Cycles engine [28] for path tracing or the Eevee engine [29] for rasterization.

After generating, the synthetic data is split into training and validation sets to train the Yolov8 model for object detection. Yolov8, chosen for its robustness and efficiency, is pre-trained on the COCO dataset. The model is then evaluated on real images using metrics like mean Average Precision (mAP), precision, and recall. These metrics are essential for accurate object detection in manufacturing, crucial for quality control and robot picking. mAP measures accuracy across object categories, while precision and recall evaluate the model ability to correctly identify and localize objects, minimizing false positives and negatives.

IV. Dataset

The pipeline was evaluated on two datasets. The first is a robotic dataset [8] includes 10 industrial 3D models and 190 manually annotated real images. Sample real and synthetic images are shown in Fig. 3 (e) and Fig. 4 (a). It provides two benchmarks: the zero-shot benchmark, where only synthetic images are used for training, and the one-shot benchmark, where synthetic images are combined with one real image for training.

The second is the SIP15-OD dataset we developed for manufacturing logistic object detection. It includes 15 3D models from three real-world manufacturing use cases, as shown in Fig. 2. Use case 1 (U1) involves five objects from a truck cabin, including black, textureless parts like the plug, airgun, electricity12v, and hook. Use case 2 (U2) involves three similar objects from the roof of a truck, with forks 1 and 2 being partially identical. Use case 3 (U3) involves seven objects from a truck transmission, all in metal material. Objects like cross, pin 1, and pin 2 have similar side profiles, while couplinghalf, gear 1, gear 2, and pinion share visual similarities. These objects illustrate common

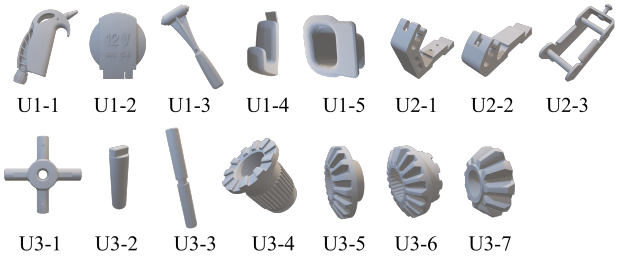


Fig. 2. 3D models of the SIP15-OD dataset. Their names are as follows in order of their number: Use case 1 (U1): 1. airgun, 2. electricity12v, 3. hammer, 4. hook, 5. plug; Use case 2 (U2): 1. fork1, 2. fork2, 3. fork3; Use case 3 (U3): 1. cross, 2. pin1, 3. pin2, 4. couplinghalf, 5. gear1, 6. gear2, 7. pinion.

TABLE I

THE NUMBER OF REAL IMAGES IN THE SIP15-OD DATASET.

	Categories	**All**	**S1**	**S2**	**S3**
U1	5	70	27	43	-
U2	3	91	27	64	-
U3	7	160	46	61	53
Total	15	321	100	168	53

industrial challenges such as textureless or metallic surfaces, geometric complexities, and subtle differences among similar categories. Samples of synthetic images generated from our pipeline for both datasets are displayed in Fig. 3 (a-d).

The SIP15-OD dataset includes 321 real images with 877 manually annotated objects, captured in varying scenarios for each use case. Scenario 1 (S1) shows parts on a conveyor belt, Scenario 2 (S2) depicts parts in blue delivery boxes, and Scenario 3 (S3) captures parts in their workstation: a black fixture or a plain surface (wooden table or padded surface) for U3. All use cases have images from S1 and S2, but only U3 includes images from S3. The dataset focuses on manufacturing logistics, where object detection models are used for verifying correct object placement for quality control or guiding robots in part-picking. Images were captured using an iPhone 12 Pro with a 12 MP camera (4032x3024 resolution) and a Poly Studio P5 webcam (2560x1440 resolution). Samples of real images are shown in Fig. 3 (e-h), with the number of real images per scenario summarized in Table I.

V. EXPERIMENTAL RESULTS

All experiments were run on A100 Tensor Core GPUs. Unless otherwise specified, all images were rendered at 720x720 resolution using the Cycles engine in Blender with path tracing. The Yolov8 models were trained with default settings from Ultralytics [11], using a batch size of 80 and an image size of 720. Rendering times using 10 Blender instances ranged from 1 to 3 seconds per image, depending on the size of 3D meshes and their texture settings. Training a Yolov8 model with 4000 synthetic images for 500 epochs took around four hours on a single GPU. Each experiment was repeated three times, and average results were reported. For all results, an IoU threshold of 0.6 and a confidence threshold of 0.5 were used.

A. Results on the robotic dataset

Ten industrial models from Horvath et al. [8] were used to generate synthetic data with our pipeline and evaluated on their real dataset. For a fair comparison, the data was trained with both Yolov4 (used by Horvath) and Yolov8. Optimal results were achieved by applying the PBR metal material texture from the CG Texture dataset [25], with other parameters randomized. As shown in Table II, our pipeline increased the mAP@50 score by more than 5% compared to Horvath et al. [8] with the same training data and model. We achieved 96.4% mAP@50 in zero-shot with Yolov8, nearly matching the one-shot model by Horvath. In their one-shot experiments, Horvath duplicated one real image 2000 times to train alongside synthetic data. However, training with synthetic data and then fine-tuning with one real image performed better in our experiments, achieving 97.5% mAP@50. Compared to Horvath, our one-shot results showed slight improvement because the zero-shot results were already high.

TABLE II

OVERALL MAP@50 SCORES (%) ON THE ROBOTIC DATASET.

Size	**Zero-shot**			**One-shot**	
	Horvath [8]	**Ours Yolov4**	**Ours Yolov8**	**Horvath [8] (2000 syn)**	**Ours**
4k	83.2	88.4	96.1	97.1	97.5
8k	83.6	90.4	96.4	-	-

TABLE III

OVERALL MAP@50 SCORES (%) ON THE SIP15-OD DATASET.

	All	**S1**	**S2**	**S3**
U1	94.1	95.4	93.3	-
U2	99.5	99.5	99.5	-
U3	95.3	92.5	95.3	95.7

B. Results on the SIP15-OD dataset

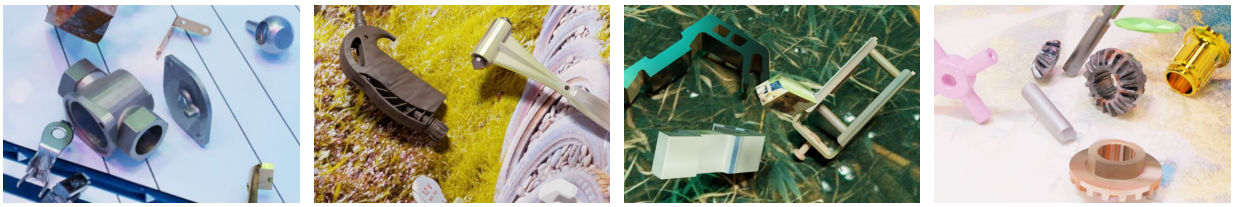
The overall results of the three use cases are summarized in Table III. All models were trained only with synthetic images and evaluated on real images, reporting the results with optimized settings. In U1, as shown in Fig. 3 (g), only one side of the objects is visible in their workstation, so we limited the rotation of 3D objects to 30 degrees in the x and y directions, generating 7,500 images in total. In U2, all parameters were randomized, generating 18,000 images. In U3, objects used metal material textures with other parameters randomized, generating 10,500 images. As shown in Table III, all use cases achieved over 90% mAP@50 across all scenarios, demonstrating the effectiveness of our pipeline.

VI. DISCUSSIONS

We conducted ablation studies to identify key factors and provide insights into the feasibility and challenges of sim-to-real manufacturing object detection with DR.

A. Key factors for DR

a. Object material properties and rendering methods. Object material properties and rendering methods are key factors for DR in manufacturing object detection. Our results surpass those of Horvath, likely due to the use of PBR metal material textures, varied illumination, and path tracing rendering. In contrast, Horvath employs material image textures and rasterization rendering. Our approach



(a) Robotics use case (b) U1 (c) U2 (d) U3

(e) Robotics use case [8] (f) U1-S1 (g) U1-S2 (h) U3-S3

Fig. 3. Samples of synthetic images from the robotic dataset and three SIP15-OD use cases (a-d). Samples of real images include (e) robotics use cases [8]. (f) S1: objects on a conveyor belt (U1 as example). (g) S2: objects in a blue delivery box (U2 as example). (h) S3: objects in their workstation (U3, placed in a black fixture). All use cases have images from S1 and S2; Only U3 has images from S3. Images are cropped for layout, maintaining original aspect ratios.

(a) Horvath et al. [8] (b) Ours (Eevee) (c) Ours (Cycles)

Fig. 4. Comparison of synthetic data rendered with different methods in the robotic use case. (a) Horvath et al. [8], using random material images textures and OpenGL rasterization. (c) Our method, using PBR metal textures and Eevee rasterization in Blender. (d) Our method, using PBR metal textures and Cycles path tracing in Blender.

TABLE IV

MAP@50 SCORES (%) COMPARISON ON DIFFERENT TEXTURES ON RENDERING METHODS.

	All textures		Realistic textures	
	R	**P**	**R**	**P**
Robotics	90.2	91.9	89.6	**96.4**
U1	92.0	**94.1**	92.4	94.0
U2	**99.5**	**99.5**	**99.5**	**99.5**
U3	94.2	94.6	94.4	**95.3**

"All textures" includes random textures (RGB, image textures, and PBR materials), while "Realistic textures" refers to PBR metal textures for Robotics and U3, PBR metal and plastic textures for U1, and PBR plastic textures for U2. "R" represents rasterization, and "P" represents path tracing rendering. The highest scores of each use case are marked in bold.

produces more realistic surface reflectance and appearance, as path tracing simulates light interactions more accurately by modeling how light bounces off surfaces, providing a more lifelike representation than rasterization. To illustrate these differences, Fig. 4 shows sample images with different textures and rendering methods from the robotic dataset. (c) generated with PBR materials and path tracing show clear visual improvements over (a) and (b), especially in handling reflective surfaces, reducing the visual gap between synthetic and real metal objects.

Furthermore, Table IV compares performance across different use cases using random textures versus realistic material textures, and rendering with rasterization (Eevee engine) versus path tracing (Cycles engine). As shown in Table IV, path tracing consistently outperforms rasterization, highlighting its importance. However, the choice of textures depends on the use case. For objects with unique materials, such as those in robotics and U3 (primarily metal), realistic PBR textures improved performance by more accurately representing real-world surfaces. For textureless objects, like those in U1 and U2 (often black and without texture), random and realistic textures yielded similar results.

TABLE V

MAP@50 SCORE (%) COMPARISON FOR PP AND D.

	Zero-shot	Without pp	Without d
Robotics	**96.4**	90.0	88.9
U1	**94.1**	92.0	90.1
U2	**99.5**	**99.5**	**99.5**
U3	**95.3**	94.4	95.1

"pp" represents post-processing, and "d" represents distractors. The highest scores of each use case are marked in bold

b. Post-processing and distractors. Post-processing and distractors generally improved model performance, as summarized in Table V. Noise and blur were applied with a 10% probability, and removing them caused performance drops in Robotics, U1, and U3, underscoring their importance. This suggests that even with extensive DR, post-processing further boosts model generalization and robustness.

Distractors, which add irrelevant objects to images, help the model handle cluttered environments, particularly improving performance in the Robotics and U1. In U2 and U3, performance remained similar regardless of post-processing and distractors, indicating their impact varies by scenario. While the improvements may not always be significant, they are generally beneficial. Based on existing research and our findings, we recommend always including post-processing and distractors when generating synthetic images.

B. Other insights for DR

Data sizes and training epochs. Object detection performance improves with increased data size up to a limit, after

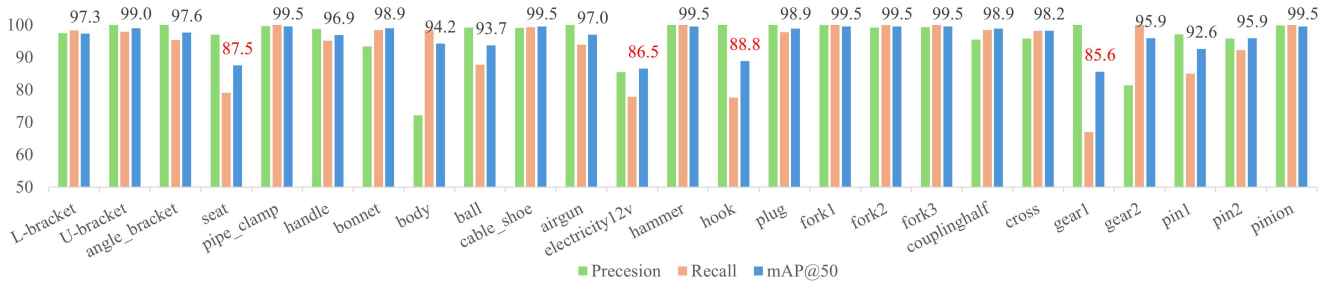


Fig. 5. Class-wise results (%) for each use case. All mAP@50 scores are shown in numbers, with scores below 90% highlighted in red.

TABLE VI

OPTIMIAL DATA SIZE (PER CATEGORY) AND TRAINING EPOCH FOR DIFFERENT USE CASES.

	All textures		Realistic textures	
	Data sizes	**Epochs**	**Data sizes**	**Epochs**
Robotics	800	500	800	500
U1	1.5k	2.5k	1.5k	1k
U2	6k	1k	3k	500
U3	3k	1k	1.5k	1k

which additional synthetic images no longer boost accuracy, consistent with previous studies. A similar effect is observed with training epochs. When training a deep learning model, performance improves with more epochs until overfitting, typically mitigated by a validation set and early stopping. However, in zero-shot training, a synthetic validation set may not reliably identify the best model for real test data due to distribution differences. Table VI presents data size limits and optimal training epochs that produced the best results. Training beyond these limits did not improve performance in our experiments. These settings were used to obtain the best results with path tracing in Table IV.

Data size limits seem influenced by the complexity of real data backgrounds. In the Robotic dataset with a controlled background (Fig. 4 (a)), 400 images per category were sufficient, with only a slight improvement (0.3% mAP@50) up to 800 images. In contrast, SIP15-OD, with more varied real-world data, required at least 1,500 images per category for strong performance. Beyond that, U2 and U3 improved slightly (about 1%) with 6K and 3K images, respectively. Specific textures helped reduce the required data size by lowering randomization in the data.

Regarding training epochs, all model performance on synthetic validation data plateaued within 500 epochs, but performance on real test data continued to improve and stabilized after 1,000 epochs for the SIP15-OD dataset. Therefore, higher early stopping thresholds can be suggested to allow the model to train longer with synthetic data, which may enhance its performance on real data.

C. Failure cases in DR

The class-wise results for all use cases are summarized in Fig. 5. Categories such as seat, electricity12v, hook, and gear1 have low recall and mAP. Two primary failure cases were identified:

a. Missed detection. Low performance is mainly due to low recall and false negatives, where objects like electricity12v, hook, and pin1 are misclassified as background. This persists even with a lowered confidence threshold, likely due to their simple geometric shapes (e.g., circles, squares, cylinders as shown in Fig. 2). During DR, with random textures and backgrounds, the model may rely heavily on shape, and these simple shapes can make it difficult for the model to use them as distinguishing features, causing challenges in learning.

b. Wrong detection. Categories like seat and gear1 perform poorly due to confusion with similar objects (e.g., seat detected as body, gear1 as gear2), as shown in Fig. 5. This is caused by their visual similarities. As illustrated in Fig. 2, gear1 and gear2 are partially identical and hard to differentiate, even in real images. In the robotic dataset, body and seat appear similar from certain angles, but they differ in scale in real images, suggesting that one-shot learning may improve the performance of the seat category.

VII. CONCLUSION AND FUTURE WORK

This paper presents a synthetic data generation pipeline for manufacturing object detection, incorporating domain randomization in object characteristics, background, illumination, camera view, and post-processing. The pipeline was evaluated on a public robotics dataset [8] and the SIP15-OD dataset we developed, which includes 15 categories across three manufacturing use cases with varied environments. Through different ablation studies, we identified key factors for generating synthetic data for manufacturing object detection: rendering with path tracing, adding post-processing, and incorporating distractors generally benefit data generation. Object material properties also play a crucial role. For unique materials like metal, structured realistic material textures are preferable to random textures. Additionally, we provide insights on optimal data sizes and training epochs, and highlight challenges in DR, such as missed and incorrect detections.

The pipeline achieved 94.1% to 99.5% mAP@50 across the four use cases, with the Yolov8 model trained only on synthetic data. These results demonstrate that the pipeline generates data that closely covers the distribution of real data, showing its robustness for various manufacturing applications like quality inspection and robotic part-picking.

However, categories with visually similar objects and simple shapes showed lower performance. Future work may focus on enhancing data generation to target these underperforming categories and improve overall detection accuracy.

REFERENCES

[1] X. Zhu, T. Bilal, P. Mårtensson, L. Hanson, M. Björkman, and A. Maki, "Towards sim-to-real industrial parts classification with synthetic dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4453–4462.

[2] L. Eversberg and J. Lambrecht, "Generating images with physics-based rendering for an industrial object detection task: Realism versus domain randomization," *Sensors*, vol. 21, no. 23, p. 7901, 2021.

[3] X. Zhu, P. Mårtensson, L. Hanson, M. Björkman, and A. Maki, "Automated assembly quality inspection by deep learning with 2d and 3d synthetic cad data," *Journal of Intelligent Manufacturing*, pp. 1–16, 2024.

[4] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. 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, pp. 23–30.

[5] J. Cohen, C. F. Crispim-Junior, C. Grange-Faivre, and L. Tougne, "CAD-based Learning for Egocentric Object Detection in Industrial Context," in *15th International Conference on Computer Vision Theory and Applications*, vol. 5. Valletta, Malta: SCITEPRESS - Science and Technology Publications, Feb. 2020, pp. 644–651. [Online]. Available: https://hal.science/hal-02553622

[6] P. Tang, Y. Guo, G. Zheng, L. Zheng, J. Pu, J. Wang, and Z. Chen, "Two-stage filtering method to improve the performance of object detection trained by synthetic dataset in heavily cluttered industry scenes," *The Visual Computer*, vol. 40, no. 3, pp. 2015–2034, 2024.

[7] I. G. B. Sampaio, L. Machaca, J. Viterbo, and J. Guérin, "A novel method for object detection using deep learning and cad models," *arXiv preprint arXiv:2102.06729*, 2021.

[8] D. Horváth, G. Erdős, Z. Istenes, T. Horváth, and S. Földi, "Object detection using sim2real domain randomization for robotic applications," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1225–1243, 2022.

[9] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield, "Structured domain randomization: Bridging the reality gap by context-aware synthetic data," in *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 20–24 2019, pp. 7249–7255.

[10] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10 781–10 790, 2020.

[11] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 213–229, 2020.

[13] X.-T. Vo and K.-H. Jo, "A review on anchor assignment and sampling heuristics in deep learning-based object detection," *Neurocomputing*, vol. 506, pp. 96–116, 2022.

[14] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, vol. 126, p. 103514, 2022.

[15] H. M. Ahmad and A. Rahimi, "Deep learning methods for object detection in smart manufacturing: A survey," *Journal of Manufacturing Systems*, vol. 64, pp. 181–196, 2022.

[16] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. Addison-Wesley, 2013.

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[20] Blender Online Community, "Blender - a 3d modelling and rendering package," 2021, blender Foundation, Blender Institute. [Online]. Available: http://www.blender.org

[21] A. S. Glassner, *An introduction to ray tracing*. Morgan Kaufmann, 1989.

[22] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: http://pybullet.org

[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[24] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of Artificial Intelligence Research*, vol. 47, pp. 853–899, 2013.

[25] L. Demes, "CC Texture Dataset," https://ambientcg.com/, 2023, accessed: 2017-02-01.

[26] J. Doe and J. Smith, "Bg-20k: A diverse background dataset for computer vision applications," *Computer Vision Journal*, vol. 34, no. 2, pp. 123–135, 2021.

[27] Blender Foundation, *Blender Python API*, 2021, accessed: 2021-09-10. [Online]. Available: https://docs.blender.org/api/current/

[28] ——, *Blender Manual: Cycles*, 2021, accessed: 2021-07-05. [Online]. Available: https://docs.blender.org/manual/en/latest/render/cycles/index.html

[29] ——, *Blender Manual: Eevee*, 2021, accessed: 2021-07-05. [Online]. Available: https://docs.blender.org/manual/en/latest/render/eevee/index.html