Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 130 (2024) 767–773

www.elsevier.com/locate/procedia

57th CIRP Conference on Manufacturing Systems 2024 (CMS 2024)

Investigating the generation of synthetic data for surface defect detection: A comparative analysis

Josefine Monnet*[a], Oliver Petrovic, Werner Herfs [a]

[a]*RWTH Aachen University - Chair of Machine Tools, Steinbachstr. 19, 52074 Aachen, Germany*

* Corresponding author. Tel.: +49 241 80-27455; fax: +49 241 80-22293. *E-mail address:* j.monnet@wzl.rwth-aachen.de

Abstract

The automatic detection of defects is a prerequisite for the optimization of processes such as sorting and reworking in all industries. However, traditional machine vision algorithms struggle with high product variance, small batch sizes, and changing environmental conditions, which has led to a shift towards AI solutions. Nonetheless, these require extensive, diverse training data, which presents hurdles in terms of acquisition and labelling. Synthetic data offers scalability and creates diverse datasets that are often difficult to obtain in the real world. They enable controlled experiments and improve the robustness of AI models by exploring different error types, sizes, orientations, and environmental conditions. However, a mismatch between simulation and reality may occur, requiring validation of applicability in the real world. For production-technical applications, it is essential to evaluate the effort of data generation. This paper compares different methods for generating synthetic data, focusing on scratches on metallic surfaces relevant to different industries. It evaluates their suitability using advanced deep learning architectures such as YOLO and DETR.

Keywords: Synthetic Data Generation; AI; Quality Inspection

1. Introduction

In modern industrial settings, the automatic detection of defects plays a pivotal role in optimizing manufacturing processes, ensuring product quality, and minimizing costs associated with rework and waste. From automotive manufacturing to electronics production, surface defect detection is indispensable for maintaining high standards and meeting customer expectations [1]. Traditional methods of defect detection, reliant on manual inspection or simplistic image processing algorithms, often fall short in addressing the complexities of modern production environments. Challenges such as product variance, small batch sizes, and fluctuating environmental conditions pose significant obstacles to the effectiveness of conventional approaches. To overcome these challenges, there has been a growing interest in leveraging artificial intelligence (AI) and deep learning techniques for defect detection. These AI-based solutions offer the promise of improved accuracy, efficiency, and adaptability to diverse manufacturing scenarios. However, their effectiveness crucially depends on the availability of high-quality training data that accurately represents real-world manufacturing conditions. Generating such training data presents its own set of challenges, particularly in industries where defects are relatively rare or difficult to capture consistently. Synthetic data generation has emerged as a promising solution to this problem, offering the ability to create diverse and realistic datasets at scale. By simulating various defect types, surface variations, and environmental conditions, synthetic data enables the training of robust AI models capable of effectively detecting defects in real-world settings [2,3,1]. In this paper, we present a comprehensive investigation into the use of synthetic data generation for surface defect detection, with a specific focus on the case of scratches on metallic surfaces. We explore various

10.1016/j.procir.2024.10.162

methods and tools for synthetic data generation, assess their effectiveness using advanced deep learning architectures, and discuss the implications of our findings for industrial quality control applications. Through this research, we aim to advance the understanding and adoption of synthetic data generation techniques in defect detection.

2. State of the art and related work

2.1. Traditional methods of surface defect detection

Quality inspection involves a systematic and structured evaluation process aimed at determining the condition of a product through examination, measurement, testing, gauging, or comparison against predefined specifications to ascertain its compliance with desired standards [4]. The reasons for using machine vision are widely recognized: It solves problems such as subjectivity, fatigue, slowness and cost associated with human inspection. Machine vision systems provide consistent, accurate and repeatable results around the clock, overcoming these challenges [5]. The idea of automated quality inspection is not new; the first applications of computer vision were developed in the semiconductor industry back in the 1970s [7,6]. Vision-based systems typically involve an algorithm trained to detect differences between the features of the inspected product and the desired features [8]. In these days, traditional computer vision approaches predominated in image processing. These conventional methods primarily relied on feature extraction from images and subsequent processing through classical algorithms. Examples of such traditional algorithms include Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), and Binary Robust Independent Elementary Features (BRIEF). These algorithms identified distinctive features in images, such as corners or edges, enabling object or pattern detection. In the case of an automated quality inspection, consistent images of the components are generated under constant conditions. Quality defects are described using such features and can then be detected automatically. The design of features in traditional computer vision approaches, however, involves a meticulous process of engineering and selection to capture meaningful characteristics from images. Engineers iterate on different feature designs, fine-tuning parameters and evaluating performance to achieve desired results [9].

2.2. Deep learning algorithms for defect detection

Due to rapid advancements in deep learning (DL) and improvements in device capabilities such as computing power, memory capacity, power consumption, image sensor resolution, and optics, there has been an increasing adoption of deep learning in various vision-based applications. Compared to previously mentioned traditional CV algorithms, DL models are trained rather than programmed, requiring less expert analysis and fine-tuning. This approach makes DL more robust against environmental factors such as changes in lighting conditions, scale and rotation, as the models learn to generalize from the data they are trained on [9]. Recent progress in machine learning techniques, notably Convolutional Neural Networks (CNNs) provides superior flexibility as they can be re-trained using custom datasets for any use case, unlike domain-specific CV algorithms [10]. In comparison to traditional rule-based approaches, however, these require a large amount of data to make them robust against previously mentioned environmental conditions [9]. Efficient automated visual defect detection using CNNs has been extensively validated [15,14,12,11,13]. Typically, researchers address the data challenge by either meticulously collecting data specific to their use case or by leveraging existing data gathered through vision systems deployed in real-world environments. However, each of these approaches results in a dataset tailored solely to their particular inspection task, lacking generalizability across different inspection scenarios. The problem of insufficient training data results on the one hand from the scarcity of available data on defective images [1], which leads to frequent bias due to the predominant number of non-defective samples [3], and on the other hand from the high amount of work involved in labeling [16]. Concerning the latter aspect, while semi-supervised learning methods have displayed encouraging outcomes in alleviating labeling efforts [17–19], the remaining issue of the scarcity of training data persists. The conventional method for improving training data involves applying geometrical transformations (e.g. rotation, flipping, shearing) to the original data samples to diversify the inputs [20] . However, this approach encounters the limitation that the improvements flatten out after a certain point as they cannot provide additional information [1].

2.3. Synthetic data generation

Generating training data through computer graphics is increasingly becoming a favored alternative [21,22]. The advantages of synthetic data generation are manifold: it allows for rapid and cost-effective production of large datasets, ensures that rare events are represented as frequently as common ones, and enables automatic pixel-level annotation of class labels. Moreover, the ability to generate training data prior to the production of the first part means that AI-powered surface defect detection becomes immediately available at the start of production. Despite promising initial research indicating the potential benefits of synthetic data [3], its widespread adoption faces significant challenges. Foremost among these is the difficulty in achieving transferability from the synthetic domain to the real domain [23]. Numerous methodologies exist for synthesizing data for visual inspection, which can be broadly classified into generative-based and rendering-based methods. Generative approaches leverage generative adversarial networks (GANs) to generate or enhance training data [20,27,25,24,26], benefiting from their inherent domain adaptation capabilities. However, a significant amount of real training data is still necessary to train GAN models effectively for generating training data of sufficient quality [1]. Despite notable progress in addressing this challenge, a state-of-the-art method for small datasets developed by researchers at NVIDIA [28] still requires over 1000 images for training. Rendering methods involve the development of simulation

models that accurately replicate the manufacturing process. Statistical variations such as changing environmental lightning conditions can be included in the training data in a controlled manner enhancing generalization capability of classifiers and ensuring consistent performance. In addition, synthetic samples can be easily regenerated to adapt to changes of environmental conditions without the need for manual data collection [23]. The use of synthetic data has been demonstrated and validated for different applications: for the inspection of structural adhesives [1] , for holes in gear forks [3] or for defects in steel workpieces [2]. The main disadvantage of this method, however, is the considerable modeling effort required to create photorealistic images [29], which must be considered in terms of use for automated quality inspection.

3. Concept

In the context of the confirmed feasibility of generating synthetic data and its subsequent use in training AI solutions, the economic viability needs to be considered regarding costs and benefits. The benefits highly depend on the chosen use case, but in any case the authenticity of the data must be guaranteed. In terms of effort, the required toolchains, which require costs for licenses and the effort involved in creating the synthetic data, must be evaluated. In order to generate an inspection solution that is as fast and low-cost as possible, the aim should be to achieve the highest possible degree of automation. The detection of scratches on metallic surfaces is chosen as an example, which represents a use case that is relevant for many companies across all industries. This use case serves as a basis for making statements on the following three aspects:

A) Feasibility: Ensuring the required authenticity of the data by checking the performance of the models on real data, identifying any challenges and best practices.

B) Toolchains: evaluation of different toolchains and their associated costs, with a focus on identifying cost drivers and assessing their impact on overall efficiency and effectiveness.

C) Automation capability: investigating the extent to which the process of data generation and training can be automated, with the aim of streamlining and scaling the overall process.

4. Implementation

4.1. Overall Toolchain

The CAD files originate directly from the engineering of the product, thereby serving as the foundation for the rendering process. The decision regarding a specific material is made as part of the product design, but may vary depending on suppliers or during manufacturing processes, which alter the surface structure through processes such as grinding or polishing. Replicating the environment is also crucial. Environmental objects that appear as such or through their shadow formation in the camera's field of view must be recreated in the simulated environment. In conjunction with this, accurately reproducing the lighting conditions is of the utmost importance. Simulation tools often have a large number of different light sources available, which are used to recreate the real conditions. Moreover, positioning the defect with a certain variability in terms of shape, size, position, and orientation on the component is essential to generate a diverse set of synthetic data, ensuring that each instance is appropriately labeled. The setup comprises a cobot within a cell, with the camera positioned at the end effector. CAD data of the cell is available and modeled in both environments. The shadow cast of the crossbeam above the cell, resulting from the ceiling light is clearly visible in Fig.1. After concluding the modelling phase, the rendering process takes place. Rendering refers to the process of generating a two-dimensional image from a three-dimensional model, encompassing the conversion of 3D model data containing geometries, textures, lighting, and shading information into pixel or vector data [30].

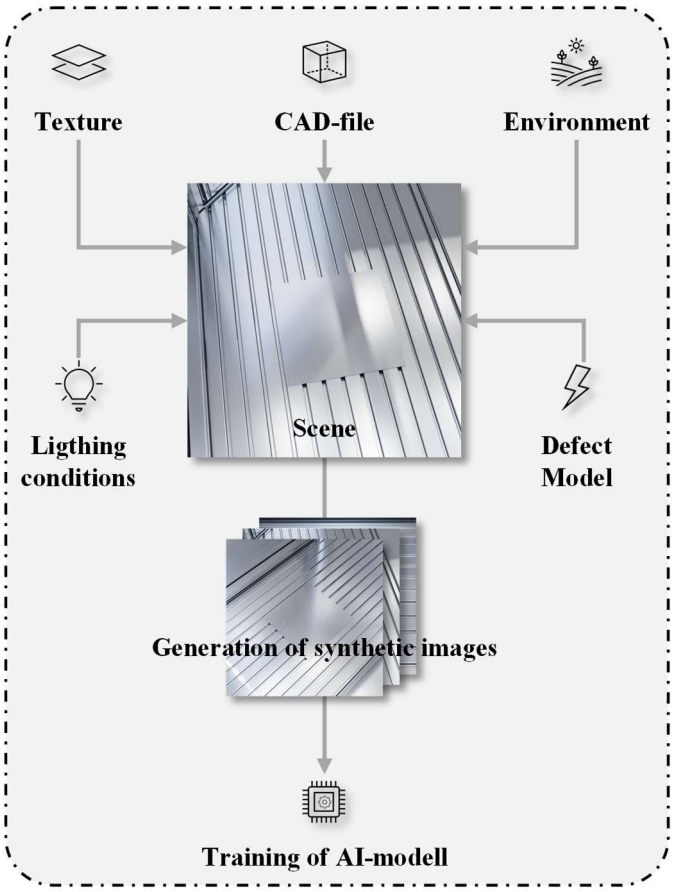

Fig. 1. Toolchain for generating synthetic data

4.2. Generation of part texture

The meticulous modeling of textures is assumed to be decisive in generating synthetic data for industrial applications. In the selected use case, surface scratches must be clearly distinguishable from the surface texture, which in this case is a rolled aluminum sheet. Simulation tools such as Blender and NVIDIA Omniverse Isaac Sim, which are used in this case, are suitable for this purpose. Within the Omniverse libraries, a variety of materials are already available in the form of Physically Based Rendering (PBR) materials, which are designed to provide a realistic representation. For the current

case, brushed aluminum was chosen, as it closely resembled real rolled aluminum.

In contrast, in Blender, the material has to be modeled using appropriate nodes. The metallic color and the reflective properties of the material were created with the combination of Color Ramps, a gradient texture and a mixer. The color of the final Color Ramp was given a spectrum in the range of light grey tones to create the base color, and a bump node was used to reproduce the fine grooves created by rolling the sheet metal. Its height parameter is fed by the noise texture of a vector distorted in the longitudinal direction.

4.3. Generation of synthetic defects

There are various options for creating synthetic scratches. In this paper, we used two different methods for creating scratches. The first approach refers to an extrusion with a correspondingly low height (<0.01 mm) applied to the base body. In this case, a straight and a curved scratch are created in a CAD program and applied accordingly. On the other hand, the effect of a scratch can be created by the use of diffuse, normal and roughness maps. The diffuse map serves as the foundation defining the base color without considering reflectivity. Meanwhile, the normal map simulates intricate surface details by altering surface normals, effectively reproducing surface irregularities. This imparts depth and texture to the rendered scratches, without physically changing the mesh. The roughness map, however, specifies how light scatters or reflects off a surface, affecting its perceived roughness. For the specific case of scratch generation, NVIDIA provides four models of scratches, including diffuse, normal, and roughness maps. Instead of utilizing these, Adobe Substance 3D Designer was employed in this instance. The creation process proceeded as follows: Firstly, a scratch shape is created using the Shape Node. Next, the scratch shape is linked to a random float variable, meaning that any changes in the randomly generated variable will result in alterations to the scratch geometry. To mimic real scratch geometry, the previously created shape is transformed into a different form using Warp and Blur Nodes. Once the desired scratch shape is achieved, the information can be outputted to the corresponding nodes.

4.4. Generation of synthetic images

To create a broad base of synthetic images, several factors must be varied. While NVIDIA Omniverse already provides a corresponding extension with pre-implemented functionality, facilitating the generation of synthetic errors, Blender requires the implementation of a script to achieve similar automation through the Blender API. Firstly, the defects themselves must vary in their characteristics such as number, rotation, length, and position. This variability ensures that the synthetic dataset encompasses a wide range of possible defects, accurately reflecting real-world scenarios. Additionally, external factors such as camera movements play a crucial role in the generation of synthetic errors. By varying parameters related to camera motion, such as angle, distance, and speed, the resulting synthetic dataset can better simulate the dynamic conditions encountered in practical settings. By carefully controlling both intrinsic error characteristics and extrinsic environmental factors, a comprehensive set of synthetic errors can be generated, facilitating robust testing and validation of detection algorithms.

Algorithm 1: Rendering of synthetic images

```
set axis rotation and camera location
FOR each camera position and orientation
  determine randomly the number of scratches
  FOR each scratch
    choose randomly between two scratch models straight/curved
    determine randomly the rotation angle around the z axis
    scale randomly in x direction
  ENDFOR
  FOR each metal plate
    get dimension and position
    FOR each scratch
      generate scratch coordinates within plate boundaries
    ENDFOR
  ENDFOR
  update camera position and rotation
  randomize lighting intensity in interval of [int_min, int_max]
  render the scene and save the generated image
  FOR each scratch
    compute scratch bounding box coordinates in camera frame
      RETURN (x_min, y_min), (x_max, y_max)
  ENDFOR
  save coordinates and delete generated objects
ENDFOR
```

Fig. 2 Pseudocode for automatic rendering of synthetic images

Subsequently, to utilize the generated images as a synthetic dataset for training a deep learning model, labeling of the applied scratches is necessary. This involves determining the coordinates of the bounding boxes surrounding the scratches. The output format is adapted to suit the requirements of the AI model, ensuring seamless integration into the training process. The render settings are also crucial for creating images that are as realistic as possible. While the default setting in Blender, the "Eevee" engine, is optimized for real-time rendering, the "Cycles" engine works according to the path tracing principle, which facilitates the computation of individual light rays (rays), enabling precise simulation of light interaction within the scene. This more computationally intensive but ensures more realistic images. To keep rendering times within acceptable limits, the number of samples per pixel was limited. Using Omniverse, the default-configuration, the RTX RealTime path tracing engine is chosen.

4.5. Selection of Deep Learning Models

In order to validate the generated training data, different deep learning models are selected and tested. On the one hand, the classical CNN-approach is chosen by testing two versions

of the YOLO-models as well as a transformer model. YOLO has emerged as a pivotal real-time object detection system, particularly in fields such as robotics, driverless cars, and video monitoring. Over time, the YOLO framework has undergone iterative improvements, progressively refining its capabilities and overcoming limitations to deliver enhanced performance in object detection tasks, the latest iteration is the YOLOv9 model [32,31]. The characteristic of the YOLO architecture is a backbone consisting of a Feed-Forward Convolutional Neural Network (CNN) coupled with a head made up of a fully connected Feed-Forward Network (FFN). The Average Precision (AP), often referred to as Mean Average Precision (mAP), serves as the standard metric for assessing the performance of object detection models. AP is calculated based on precision-recall metrics, accommodating multiple object categories and defining positive predictions using Intersection over Union (IoU). Precision determines the accuracy of positive predictions, while recall measures the proportion of actual positive cases correctly identified by the model. There is typically a trade-off between those two [1,32]. Since YOLOv3, this metric has been evaluated using the Microsoft COCO (Common Objects in Context) dataset [33]. In January 2023, Ultralytics unveiled YOLOv8 [34], aligning with the prevailing trend towards anchor-free models. The latest YOLO model, YOLOv9, was improved by the application of the Programmable Gradient Information (PGI) concept, reducing information loss in deep neural networks during forward propagation and enhancing model updating. Additionally, the Generalized Efficient Layer Aggregation Network (GELAN) architecture offers potential improvements in YOLOv9's efficiency, maximizing parameter utilization while maintaining lightness, speed, and accuracy [31]. Due to the trade-offs between speed and accuracy that have emerged during the development of the framework, there are different sizes for each model generation of YOLO [32]. The DEtection TRansformer model (DETR) is a novel architecture for object detection, trained on the COCO 2017 dataset, and it has shown success in text and audio processing. It employs an Encoder-Decoder Transformer with a CNN backbone, utilizing Object Queries to search for specific objects in images. Each Object Query passes through the decoder, providing both class predictions and bounding box predictions for the corresponding object [35]. As can be seen in Table 1, YOLOv9C outperforms the other two models in terms of mAP, but is more computationally intensive for predicting results.

Table 1. Comparison of different model architectures [37,34,36]

	YOLOv8m	YOLOv9C	DETR
mAP (COCO dataset)	50,2 %	53,0 %	42,0%
Required GFLOPs	78,9	102,1	86

5. Results and Evaluation

The following presents the results of the prediction, enabling an assessment of the authenticity of the generated data. Furthermore, the tested toolchains are compared, and a statement regarding the automation capability of the process of generating synthetic data is made.

5.1. Data authenticity

For the selection of the models, all three models were trained with the synthetic data and their results, both after mAP and after the required training time (using a Tesla T4, 16 GB VRAM), were compared; the results can be seen in Table 2. Similar to the COCO dataset, the YOLO models outperform the transformer model. Due to the significantly higher accuracy, the YOLOv9 model is selected for further investigations.

Table 2. Results of different model architectures on synthetic dataset

	YOLOv8m	YOLOv9C	DETR
mAP (syntetic scratch dataset)	46,3 %	66,2 %	32,8 %
Required training time (min)	38.65	59.74	159,36

To assess the authenticity of synthetic data adequately, it is essential to evaluate the performance of the model. This evaluation involves the use of two distinct datasets for training. Dataset 1 consists entirely of synthetic data, comprising 435 images, each depicting a different number of scratches. Dataset 2, on the other hand, includes an additional 15% of real images sourced from four plates. These real images vary in the number of scratches applied and the camera position. As can be seen in Figure 3, the integration of a small amount of real data significantly improves the prediction result. In the purely synthetic training data set (c), the edges of the T-slotted plate are recognized as defects. This happens even though it was modeled in the simulation. This could possibly be due to the difference between the simulated and real lighting conditions.

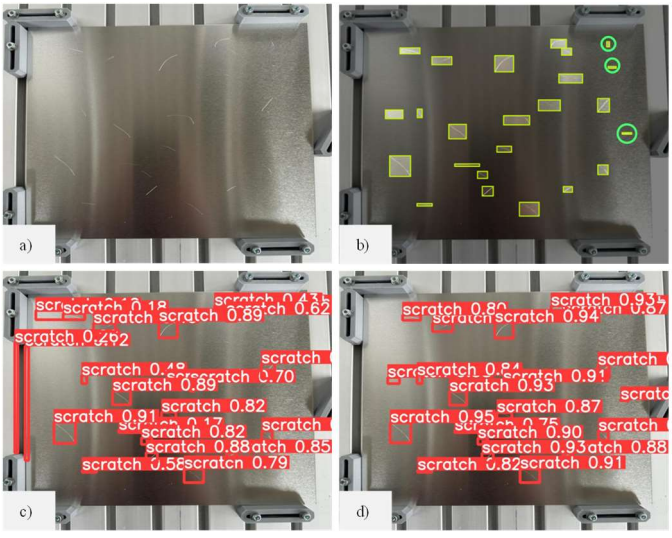

Fig. 3. Results of prediction on real data
(a) image of the aluminum sheet with scratches (b) annotated image
(c) Dataset 1: TP: 19, FP: 3, FN:5; Precision: 79,2%, Recall: 86,3%
(d) Dataset 2: TP:21, FP:0, FN:3; Precision: 87,5%, Recall 100%

The T-slots appear significantly brighter in the simulated images than in the real images. On the second data set (d), no scratches are detected by mistake (FP:0), but three of the total

of 24 scratches are not detected. By looking closely to these scratches (a)+(b), they are very fine scratches that are difficult to recognize from the current perspective. It can be assumed that even better results could be achieved here by tuning the lighting and camera angle. In summary, it can be said that the authenticity of the generated data is sufficient to recognize scratches in real images.

5.2. Toolchains

In the present use case, surface texture generation proved to be straightforward in Isaac Sim, benefiting from the availability of a closely matching material within the Omniverse library. However, in Blender, material properties had to be constructed using nodes. Blender boasts a vast community, facilitating access to a multitude of tutorials that yield rapid and satisfactory outcomes. Conversely, documentation for material nodes in Omniverse, serving a comparable purpose, is notably scarce. Consequently, crafting or adapting materials "from scratch" within this framework entails a considerable degree of trial and error. The generation of synthetic defect images, in this case scratches, was carried out through two different approaches. Firstly, CAD models of scratches were created, and secondly, normal, roughness, and diffuse maps were generated. In the context of a production-oriented application, it is reasonable to assume that companies possess both the required licenses and the necessary expertise to generate these defects using the respective CAD programs. However, this may not necessarily be the case for the generation of normal, roughness, and diffuse maps. Nevertheless, this approach is significantly less computationally intensive. The assessment of which method is more suitable likely depends heavily on the specific use case. The tuning of rendering settings, which is necessary in Blender to produce realistic images, was unnecessary in Omniverse, as very realistic images were already generated by default. Within the NVIDIA Omniverse framework, there is a plugin available in Isaac Sim that facilitates the selection of components, synthetic defects in the form of normal, diffuse, and roughness maps, as well as intervals for parameter randomization via a user interface. This functionality can be extended by modifying the underlying Python script. This makes it very user-friendly at first glance. However, despite numerous tests and attempts to use the plugin without modification or with controlled adjustments to fit specific needs, effects such as black screens, crashes, and "Out-of-Memory" errors occur repeatedly. In addition, customizing the system to address specific issues is challenging due to unclear documentation. Omniverse and other Nvidia rendering products can only be used with Nvidia graphics cards. In Blender, these functionalities need to be implemented directly through the Python API, which was easily achievable, allowing seamless customization and extension of the corresponding features. Despite the necessary fine-tuning of rendering methodology and texture selection required to produce realistic images with Blender , it offers the capability to randomize scenes and output bounding box data relevant to image recognition directly and in a user-defined manner through custom scripts. Additionally, Blender benefits from better documentation compared to Omniverse. Particularly in the context of further automating the toolchain, the application-specific customization holds special significance.

5.3. Automation capability

The part currently automated involves the randomization of camera perspectives as well as defect- and lighting-related parameters. However, the highest engineering effort arises during the creation of textures. In terms of applicability to manufacturing, different approaches are possible. The concept of databases containing textures presents a straightforward solution. In addition, the use of GenAI, which is only used to generate the textures, would also be conceivable here. These strategies could significantly streamline the process of texture, paving the way for more efficient and scalable manufacturing applications. However, the use of photometric stereo techniques to capture textures and the subsequent manual projection onto the objects [3] appears comparatively complex compared to the other approaches presented. Regarding the generation of defects, two different methods were presented preferring the use of normal, diffuse and roughness maps. Once created, they can be used for other parts. If other effects such as paint damage or rust are discovered, these can be modeled in a similar way. Tools such as Blender or Adobe Substance 3D Designer are well suited due to their good documentation and large community.

6. Conclusion and further work

This study delved into synthetic data generation for surface defect detection, focusing on scratches on metallic surfaces relevant to various industries. As AI-based defect detection demands extensive training data, synthetic data emerges as a scalable solution, offering diverse datasets not easily obtained in reality. By simulating various defect types and conditions, synthetic data facilitates robust AI model training. However, ensuring synthetic data's applicability to real-world scenarios requires validation. Our research compared different synthetic data generation methods and evaluated their effectiveness using advanced deep learning architectures. While user-friendly tools like NVIDIA Omniverse offer ease of use, they suffer from stability issues and lack clear documentation. In contrast, Blender provides greater customization and automation potential, though requiring fine-tuning for realistic image generation. We also emphasized the importance of automating the toolchain for efficient defect generation. Leveraging automated texture synthesis techniques by using GenAI approaches could streamline texture and defect creation, enhancing manufacturing efficiency and scalability.

Acknowledgements

The IGF-projekt 22648 N/2 (ROOKIE) of the research association FVP was supported via the AiF within the funding program “Industrielle Gemeinschaftsforschung und -entwicklung (IGF)” by the Federal Ministry for Economic Affairs and Climate Action (BMWK) due to a decision of the German Parliament.

References

[1] Silva Peres, R., Guedes, M., Miranda, F., Barata, J., 2021. Simulation-Based Data Augmentation for …Structural Adhesive With Deep Learning. IEEE Access.

[2] Boikov, A., Payor, V., Savelev, R., Kolesnikov, A., 2021. Synthetic Data Generation for Steel Defect Detection and Classification Using Deep Learning. Symmetry 13 (7), 1176.
[3] Gutierrez, P., Luschkova, M., Cordier, A., Shukor, M., Schappert, M., Dahmen, T., 2021. Synthetic training data generation for deep learning based quality inspection, 13.
[4] Winchell, W., 1996. Inspection and Measurement in Manufacturing: Keys to Process Planning and Improvement;. Society of Manufacturing.
[5] El Azab, W., Cousin, S., 2021. VisualInspectionPracticesofCleanedEquipment_PartI.
[6] Jarvis, J.F., 1980. A method for automating the visual inspection of printed wiring boards. IEEE transactions on pattern analysis and machine intelligence 2 (1), 77–82.
[7] Harlow, C.A., Henderson, S.E., Rayfield, D.A., Johnston, R.J., Dwyer, S.J., 1975. Automated Inspection of Electronic Assemblies. Computer 8 (4), 36–45.
[8] Sarvesh Sundaram and Abe Zeid, 2023. Artificial Intelligence-Based Smart Quality Inspection for Manufacturing.
[9] Mahony, N., Campbell, S., Carvalho, A., Harapanhalli, S., Velasco Hernandez, G., Krapalkova, L., Riordan, D., Walsh, J., 2020. Deep Learning vs. Traditional Computer Vision.
[10] Schmedemann, O., Baaß, M., Schoepflin D., Schüppstuhl, T., 2022. Procedural synthetic training data generation for AI-based defect detection in industrial surface inspection.
[11] Staar, B., Lütjen, M., Freitag, M., 2019. Anomaly detection with convolutional neural networks for industrial surface inspection. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering.
[12] Soukup, D., Huber-Mörk, R., 2014. Convolutional Neural Networks for Steel Surface Defect Detection from Photometric Stereo Images.
[13] Weimer, D., Scholz-Reiter, B., Shpitalni, M., 2016. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. CIRp.
[14] Kim, S., Kim, W., Noh, Y., Park, F., 2017. Transfer learning for automated optical inspection, 2517–2524.
[15] Faghih-Roohi, S., Hajizadeh, S., Nunez, A., Babuska, R., Schutter, B. de, 2016. Deep convolutional neural networks for detection of rail surface defects. International Joint Conference on Neural Networks, 2584–2589.
[16] He, Y., Song, K., Meng, Q., Yan, Y., 2020. An End-to-End Steel Surface Defect Detection Approach via Fusing Multiple Hierarchical Features. IEEE Trans. Instrum. Meas. 69 (4), 1493–1504.
[17] Dai, Z., Yang, Z., Yang, F., Cohen, W.W., Salakhutdinov, R., 2017. Good Semi-supervised Learning that Requires a Bad GAN. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
[18] Di, H., Ke, X., Peng, Z., Dongdong, Z., 2019. Surface defect classification of steels with a new semi-supervised learning method. Optics and Lasers in Engineering 117, 40–48.
[19] Kumar, A., Sattigeri, P., Fletcher, P.T., 2017. Semi-supervised Learning with GANs: Manifold Invariance with Improved Inference. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
[20] Jain, S., Seth, G., Paruthi, A., Soni, U., Kumar, G., 2022. Synthetic data augmentation for surface defect detection and classification using deep learning. J Intell Manuf 33 (4), 1007–1020.
[21] Peng, X., Sun, B., Ali, K., Saenko, K., 2015. Learning Deep Object Detectors from 3D Models.
[22] Su, H., Qi, C.R., Li, Y., Guibas, L., 2015. Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views.
[23] Retzlaff, M., Richter, M., Längle, T., Beyerer, J., Dachsbacher, C., 2016. Combining synthetic image acquisition and machine learning: accelerated design and deployment of sorting systems.
[24] Singh, R., Garg, R., Patel, N., Braun, M. Generative Adversarial Networks for Synthetic Defect Generation in Assembly and Test Manufacturing.
[25] Niu, S., Li, B., Wang, X., Lin, H., 2020. Defect Image Sample Generation With GAN for Improving Defect Recognition. IEEE Trans. Automat. Sci. Eng., 1–12.
[26] Zhang, G., Cui, K., Hung, T.-Y., Lu, S. Defect-GAN: High-Fidelity Defect Synthesis for Automated Defect Inspection.
[27] Meister, S., Möller, N., Stüve, J., Groves, R.M., 2021. Synthetic image data augmentation for fibre layup inspection processes: Techniques to enhance the data set. J Intell Manuf 32 (6), 1767–1789.
[28] Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., Aila, T., 2020. Training Generative Adversarial Networks with Limited Data.
[29] Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., Birchfield, S., 2018. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects.
[30] Lehn, K., Gotzes, M., Klawonn, F., 2023. Introduction to Computer Graphics.
[31] Wang, C.-Y., Yeh, I.-H., Liao, H.-Y.M., 2024. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information.
[32] Terven, J., Cordova-Esparza, D., 2023. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. MAKE 5 (4), 1680–1716.
[33] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P., 2015. Microsoft COCO: Common Objects in Context.
[34] Ultralytics. YOLOv8. https://github.com/ultralytics/ultralytics.
[35] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S., 2020. End-to-End Object Detection with Transformers.
[36] Ultralytics. YOLOv9. https://docs.ultralytics.com/models/yolov9/#performance-on-ms-coco-dataset. Accessed 10.04.24.
[37] FaceBookResearch. DETR. https://github.com/facebookresearch/detr. Accessed 10.04.