Investigation of an integrated synthetic dataset generation workflow for computer vision applications

Julian Rolf [0000-0002-3215-9265], Mario Wolf [0000-0002-0628-7570], Detlef Gerhard [0000-0002-3266-7526]

Ruhr-Universität Bochum, Germany
julian.rolf@ruhr-uni-bochum.de

Abstract. Object detection and other machine learning technology applications play an important role in various areas of computer vision (CV) applications within the product lifecycle, especially in quality assurance or general assembly assistance. While the implemented CV-based systems provide great benefits, training and implementing deep learning models is often a tedious and time-consuming task, especially in the field of object detection. To accomplish good results, large datasets with a high quantity of object instances in a bright variety of poses are required These are generally created manually and are therefore very time consuming to create.
To improve the training process, synthetic training data can be used. It is generated within a virtual environment using a product's geometry model. In this paper, the authors propose a synthetic dataset generator for object detection, that is integrated into a PLM system to automate the process of collecting and processing the CAD data for creating the synthetic machine learning training dataset. Domain randomization is used to eliminate the effort of creating a virtual environment, to fully automate the dataset generation, and to increase the generalization of the model. The trained detector is tested on an object detection demonstrator set-up to evaluate its performance in a real-world use case. For evaluation purposes, the authors also provide a comparison of the test results to an object detection model that is trained without domain randomization, using a very close-to-reality virtual environment.

Keywords: deep learning, synthetic data generation, PLM integration, object detection, computer vision.

1 Introduction

The demand for increasing degrees of automation in different parts of the product life cycle inevitably leads to an increase in the complexity of the required assistance systems. In many areas machine learning (ML) can fulfill this need completely or partially. ML has an enormous potential for the support of assembly and disassembly processes, quality assurance, or in the improvement of product traceability, especially when using computer vision (CV) techniques. However, the implementation of ML systems is associated with high initial effort, particularly for the ML training process. A large amount of suitable training data is necessary for CV applications to work

reliably. It is therefore necessary to create new data sets for the detection of each new individual product. This includes not only the creation of images of the respective products, but also the correct and precise labeling of the collected images. This process is not only associated with an immense amount of work but is also error-prone and literally impossible to obtain precise results manually, as is the case with the segmentation of image objects. To solve these problems, the approach is to use synthetic data. The term ‘synthetic data’ refers to information that does not originate from real sources. In CV, for example, a renderer is used to generate images of the desired objects. Information about position and orientation of the objects is known to the rendering engine and can therefore be added automatically to a generated data set for labelling purposes in supervised machine learning approaches. In this way, in a comparatively short time frame, any amount of data can be generated and used for training a neural network. The only prerequisite for generating the data is the availability of 3D representations of the objects of interest and their accuracy in depicting the (physical) objects that are to be detected later in the process. PLM systems are therefore ideally suited as a data source for the synthetic data generation process.

This is the first paper of a total of three. As presented in figure 1. the overall vision is to build an assistant assembly workstation, using deep learning algorithms to assist the assembly process. The workstation will be set up automatically, by generating the needed training data, based on the 3D representations of the objects to be detected. Further analysis on the parts will be conducted to increase the performance of the system. This includes grouping similar and ignoring smaller objects. In this paper, the authors focus on the first aspect and present a data generation pipeline to fully automate the process of generating synthetic data for object detection. Additionally, the data generator is connected to a PLM system to demonstrate the integration capabilities of the proposed solution with existing systems and workflows.

Fig. 1. Overview of the three aspects to be covered.

2 Related Work

This chapter covers use cases for CV applications in various parts of the product lifecycle. Additionally, we will cover modern synthetic data generation tools.

2.1 Computer Vision

In (dis)assembly processes, the correct identification and localization of components is a necessary requirement for the automation of work steps. For the removal of screws in a PCB with a robotic arm, Mangold et. al used an object detector, which made it possible to determine the correct tool for different types of screws [1]. Therefore, performing the required work steps could be fully automated. Another screw detection detector for robot disassembly was implemented by Brogan et. al [2]. CV algorithms also play a major role in quality control. Basamakis et. al used an object detector to determine missing, correctly or incorrectly assembled rivets and achieved a total accuracy of 83% [3].

2.2 Synthetic Data

A variety of options exist for generating synthetic data for CV use cases, such as the Perception tool for the Unity 3D Engine [4]. Another generator is Nvidia's Omniverse Replicator, based on Nvidia's Omniverse platform [5]. It can create graphically detailed data through the native use of ray tracing. In 2022, Google Research introduced the Kubric data generator [6]. It is based on the Blender render engine and uses the Bullet physics simulation. With SynthAI [7], Siemens offers a cloud service for the generation of synthetic data based on CAD files.

Synthetic data is particularly useful when not enough data can be collected through classical methods to ensure sufficiently good performance of the trained AI model. In addition, it often requires less effort to create a suitable data set and is therefore more cost-effective. In consequence, some datasets have synthetic twins to increase the total amount of training data, like the virtual KITTI dataset [8]. Another way of creating synthetic data is by using a Generative Adversarial Network (GAN). This particular type of deep learning network can be trained on smaller real datasets and can then be used to augment the dataset with synthetic data to produce a sufficiently large amount of training samples [9].

3 Aims and Requirements

In the context of the presented research work, it is investigated how to integrate the synthetic data generation process in a PLM system for full automation. This enables the use of ML while keeping the required expert knowledge in the field of ML very low. The PLM system takes the role of the data source, workflow engine, and, after completion of the data generation, the role of the data provider for the ML application. The generated data set is used for training an object detector to check the quality of the data set. For this purpose, the object detector is tested with a dataset containing real images, which were labeled manually. In addition, a second synthetic data set is generated manually, which represents the properties of the test environment as accurately as possible. This allows a quality comparison of the two detectors and a classification of the practicability of using the general data set.

The goal of this work is the development and validation of a concept for the automatic generation of a synthetic data set for CV problems, in connection with a PLM system for the management of the detected objects. The data set generation process is to be started and monitored from within the PLM system. The generated data set should support the most common CV problems, such as 2D/3D object detection, object segmentation or depth estimation. In addition, there may be little to no knowledge about the real-world application domain of the ML model at the time of training data generation. Therefore, there is a requirement for general applicability of the dataset, which will involve domain randomization.

4 Concept

In this chapter, the concept is presented and explained in detail Firstly, the data set generator to create the CV data set based on a general service interface. Secondly, we elaborate on a PLM system integration that manages the work process as well as the parts and components involved. The last section of this chapter deals with a connector for the communication of the PLM tenant with the dataset generator.

4.1 Dataset Generator

The dataset generator is responsible for generating and creating the CV dataset independently from other components. The core of the generator is the render engine, which is used to create images from a virtual 3D scene. The properties of the scene, such as background, lighting and shadows, as well as the orientation and position of the objects in the scene, depend on the parametrization. The choice of these properties plays a decisive role in the performance of the model trained on this data set. If crucial features of the test environment or of the objects to be recognized are missing, the model is limited in its ability to correctly interpret the test inputs. However, the development of a data set adapted to the test environment requires previous domain knowledge that then influences the test environment setup. To solve this problem, the concept of domain randomization is used [10]. Here, the training environment is randomized in as many properties as possible and feasible. This increases the variance of the training data and provides a better generalization of the model. In this way, a generally usable data set can be generated, which can additionally be extended with a lesser amount of training data with high domain knowledge about the test environment to further increase the performance of the model in said environment. In the *Outcomes* chapter, the performance of an object detector trained with a synthetic dataset based on the presented concept is discussed in more detail, as well as the general usability of such a dataset.

The structure of the virtual 3D scene consists of a dome, a hemisphere with a bottom plate. A High Dynamic Range Image (HDRI) texture is placed on the inside of the geometric body. This texture contains not only a texture, but also information about the light properties of the scene. Inside the dome, an arbitrary number of objects, which are to be identified later in the process, are placed, with random orientation and position.

Finally, a virtual camera is placed and aimed at the center of the floor at a random height inside a defined spectrum. To further increase randomization, the color of the objects can also be varied. Since the position and orientation of the objects are precisely known in the virtual space, annotation information, such as bounding boxes or segmentation, can be automatically generated.

In order to be able to interact with other services, such as PLM systems, the generator provides a RESTful webservice interface. An endpoint is used to place a request for data set generation. The jobs are processed according to the 'First In - First Out' (FIFO) principle. A job consists of several settings or parameters, that are stated in Table 1.

The Webhook mail address defines an optional URL. If the dataset generation status changes, a POST request with the corresponding status is sent as payload to the defined URL. This way the generator can inform about the start and the completion of the generation. Also, errors that occur during the generation can be forwarded. The return of the POST request is a UID for the unique identification of the order.
The finished data record can be downloaded as an archive via a second endpoint. In addition, the current status of the order can also be queried directly via a third endpoint. For the last two endpoints, the order UID is required.

4.2 Connector

One challenge in designing the interface of the generator is the very different approach between current PLM systems regarding interface connectivity. These vary greatly in their range of functions, such as handling with Webhooks and documents or 3D representations. Depending on the system, interaction with the interface of the data set generator is therefore not possible without restrictions. To solve this problem, a connector is designed. The connector is a micro service and is located as the middle man of the communication between the generator and the PLM system. Its task is to override or adapt the communication, so that a correct and successful data exchange can be guaranteed and the use of the full functionality of the PLM system can be ensured. Since the connectors are adapted to one system at a time, a connector must also be developed for each supported PLM system. The scope of the connector and thus also the implementation effort depends on the particular PLM system used.

Table 1. Dataset creation parameters with default, min and max values

Parameter	Default Value	Min. Value	Max. Value
Dataset Size	5	1	1000
Resolution	256 x 256	64 x 64	512 x 512
Scale	1.0	0.01	100.0
Color	Uniform Sample		
Obj. Min. Occurrences	1	0	Obj. Max. Occurrences
Obj. Max Occurrences	5	Obj. Min. Occurrences	
Webhook URL	Not set		

4.3 PLM-System

A PLM system is used to manage product relevant data, in particular the parts to be detected, including their 3D representations and all other. In addition, the data generation workflow of the respective item is started and monitored by the PLM system via the connector. The workflow is linked to a single object, so that a generated data set always represents exactly one object. The workflow consists of five states and is displayed in figure 2.

The initial state is the start state of the workflow and describes that a data generation job can be created. If a data generation job is created, the availability of the required 3D representation of the selected object as an attachment or document is checked. If the availability is given, the adapter is informed about the existence of the new order, which forwards it to the generator. In addition, the connector registers with the generator via Webhook to be informed about events. In this way, the connector can trigger the correct transition to the current state in the PLM system after receiving the information. If the generation is completed, a mail including the download link of the generator is created and sent to the owner of the respective object in the PLM system.

For the transitions to be triggered, a user with the appropriate rights profile is created and managed by the connector or used by it.

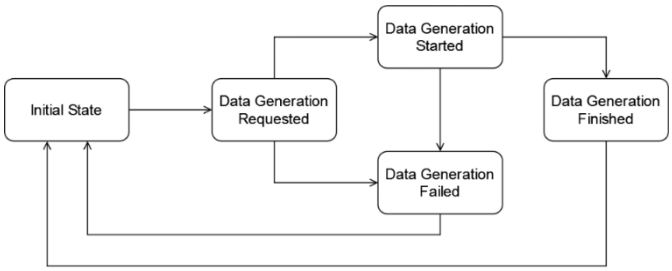

Fig. 2. Data generation workflow

5 Implementation

The implementation of the data generator is based on the data generation pipeline Kubric [6] from Google Research. It is able to generate a variety of different types of CV data sets, like segmentation, bounding box or depth estimation. Kubric uses Blender as the render engine. Also, PyBullet is used as physics engine to simulate physically correct collisions of objects, if desired. This way, more realistic data can be generated, but it also requires more complex processing of the objects, due to the generation of a collision mesh, and is therefore not used in this implementation. The objects are instead randomly placed and oriented in the scene, within an area of 0.3m x 0.3m around the origin at a height of 0.01m. This ensures that the objects are captured by the camera. HDRI textures for the background and lighting of the scene are provided by Kubric via an API from HDRI-Heaven and placed on the inside of the dome. A total of up to five

shots are taken per scene. The positions and orientations of the objects vary in each shot. However, the texture of the background remains the same for each scene.

The data generation takes place in a worker sub-process. The main process hosts the REST API. Jobs are managed in a FIFO data structure and processed accordingly. The Kubric process uses the webhook URL to inform if a job finished successfully or with errors. The IDs of successful jobs are managed in a separate data structure, so that the API can make these records available for download. Figure 3 shows this process sequence graphically. In addition, records older than 24 hours are deleted by a third process to optimize the application's memory consumption.

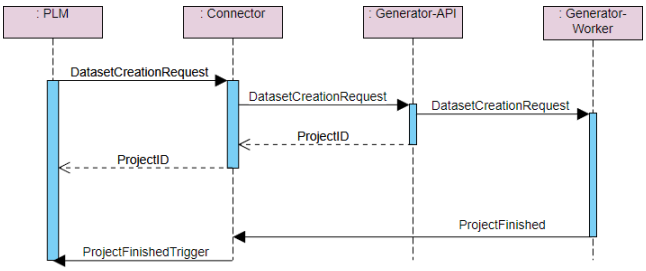

Fig. 3. Sequence Diagram of the dataset generation workflow

For the prototype implementation and evaluation purposes, Autodesk Fusion Manage 360 is used as the PLM system. A workspace in Fusion Mange is created, including a workflow as described in the concept. During the transition from the "Initial state" to the "Data Generation Requested" state, a script is called that searches for an OBJ file with the same name as the object in the attached documents in the Workspace. Thereupon a POST request is sent to the connector. This contains the parameters described in the concept for the data set generation and additionally the WorkspaceID, ItemID and TransitionsIDs of the PLM system, so that the transitions to be switched can be identified and triggered by the connector. The comment of a transition is read in by the script. This way the user is able to change the parameters for the dataset generation. The ClientID and the Client-Secret of a user with the corresponding rights profile for switching the transitions are stored in the connector.

6 Outcomes

In order to validate the general usability of a generated data set, a test procedure is developed and carried out. In total, three reality-based data sets are created. The first one consisting of 50 photos and the second one consisting of 20 photos, for 2D bounding Box detection, are created with a single object class. The third dataset also contains 50 samples but has five different object classes, instead of just one. The real scene used has good lighting conditions and a monochrome gray tabletop as background, to mimic a typical assembly workstation, but without any distracting objects. Light and background remain unchanged during the acquisition of the datasets. Only the number, orientation and position of the objects differ. The first and third one do not contain any

overlapping bounding boxes and have a sparser object arrangement. The second reality-based data set contains a very tight arrangement of the objects, including overlapping bounding boxes. All three data sets are used to test three different object detectors trained with synthetic data. Detector A is trained on a data set created with the help of the data generator. Accordingly, many properties of the virtual scene are randomized. This also applies to the color of the objects. Detector B is trained on a synthetic dataset, which is strongly adapted to the test environment, using Unity3D. In this dataset, the background, the lighting conditions, and the color of the objects match the real environment. Likewise, the objects were placed physically correctly on the background. Both training datasets only have a single object class. For the third test dataset with multiple classes, a third detector is trained using a third train dataset with synthetic images generated by the proposed data generator.

The first two training data sets each consists of 1000 images. The third one consists of 5000 images. 1000 images for each object class. Currently the data-generator is only able to handle one object at a time. To be able to train multiple classes, one dataset for each object is generated. Afterwards the datasets are combined. Figure 4 shows a comparison of images from the three synthetic training datasets and the three real test datasets.

Fig. 4. Samples of the synth. training data in the first row (left and middle one generated by the data-generator, right one with Unity3D) and the real test data in the second row (left sparse, middle tight and right mul. class dataset)

The time needed to generate a single synthetic dataset for one object class with the implemented generator takes approximately 45 minutes for 1000 samples. The generation time is not only depended on the available hardware and the applied resolution, but it also depends on the images rendered per scene. With five images rendered each scene, half of the total generation time is conducted for creating the virtual blender environment. Increasing this value would therefore lower the generation time. All models were trained for a total of 300 epochs, using the small version of the yolov5 object detection model [11].

Table 2. Test results for all three detectors

Model	Dataset	Precision	Recall	mAP50
Detector A	sparse	0.980	0.974	0.982
Detector B	sparse	1.000	0.993	0.995
Detector A	tight	0.211	0.085	0.077
Detector B	tight	0.918	0.785	0.851
Detector C	mul. classes	0.800	0.937	0.911

Precision is a percentage value that relates the amount of correctly identified objects to the amount of all objects supposedly found by the model. Recall is the amount of correctly identified objects relative to the number of objects that should have been detected. Mean Average Precision (mAP) also considers the Intersection of Union of the bounding boxes. For mAP50, the IoU ranges from 50% to 95%, with a step size of 5%.

As shown in Table 2. detectors A and B achieved near perfect results on the first dataset and can identify and localize the objects within the image with only a hand full of false detections. The better performance of model B is expected, due to the training data adapted to the test environment but is only slightly pronounced. The tests on the second dataset reveal a huge performance gap between the two detectors. The mean average performance of detector A is eleven times better than that of detector B.
The third detector, trained and tested on multiple classes does not perform as perfect as detector A and B on the sparse dataset, but is still able to correctly detect the objects in most cases.

7 Conclusion

In conclusion, the test procedure validates the usability of the generated data set for training an object detector for simple structured environments. This includes environments without distractions, such as objects not to be detected, with optimal lighting conditions and a rather sparse arrangement of the objects. In such a case, the comparison between model A and B shows, that dealing with the effort of adapting the training dataset to the test domain, or even creating a training dataset manually, is not worth the effort and can be automated by using domain randomization. While the data-generation pipeline currently only supports one object per dataset, combining the generated datasets for multiple class problems is a feasible approach for the number of classes tested. In a more complex environment, the detector A trained on the randomized synthetic data fails to reliably identify and localize the individual objects, while detector B still performs reasonably well. Accordingly, the cause of the breakdown in the performance of detector A is to be found in the randomized synthetic data set and the randomized properties of the virtual scene. In order to increase the robustness of a detector trained on a randomized data set, tight arrangements of the objects in the virtual scene can be

made. A subdivision of the data set into different groups based on edge cases to be mapped is conceivable.

The authors were able to demonstrate that an end-to-end process for the generation of synthetic datasets for CV problems can be fully automated and linked to a PLM system. The required expertise in the field of machine learning for the creation of such a dataset is very low and can therefore be well integrated into the PLM environment.

References

1. Mangold S, Steiner C, Friedmann M, Fleischer J (2022) Vision-Based Screw Head Detection for Automated Disassembly for Remanufacturing. Procedia CIRP 105:1–6. https://doi.org/10.1016/j.procir.2022.02.001
2. Brogan DP, DiFilippo NM, Jouaneh MK (2021) Deep learning computer vision for robotic disassembly and servicing applications. Array 12:100094. https://doi.org/10.1016/j.array.2021.100094
3. Basamakis FP, Bavelos AC, Dimosthenopoulos D, Papavasileiou A, Makris S (2022) Deep object detection framework for automated quality inspection in assembly operations. Procedia CIRP 115:166–171. https://doi.org/10.1016/j.procir.2022.10.068
4. Borkman S, Crespi A, Dhakad S, Ganguly S, Hogins J, Jhang Y-C, Kamalzadeh M, Li B, Leal S, Parisi P, Romero C, Smith W, Thaman A, Warren S, Yadav N (2021) Unity Perception: Generate Synthetic Data for Computer Vision. http://arxiv.org/pdf/2107.04259v2
5. NVIDIA Omniverse Replicator, https://developer.nvidia.com/nvidia-omniverse-platform/replicator, last accessed 2023/01/25.
6. Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, Andrea Tagliasacchi (2022) Kubric: a scalable dataset generator
7. Siemens SynthAI, https://synth.ai.sws.siemens.com/, last accessed 2023/01/25.
8. Cabon Y, Murray N, Humenberger M (2020) Virtual KITTI 2. https://arxiv.org/pdf/2001.10773
9. Frid-Adar M, Klang E, Amitai M, Goldberger J, Greenspan H (2018) Synthetic data augmentation using GAN for improved liver lesion classification. In: 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), pp 289–293
10. Tobin J, Fong R, Ray A, Schneider J, Zaremba W, Abbeel P (2017) Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. https://arxiv.org/pdf/1703.06907
11. Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu, Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, Mrinal Jain (2022) ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. Zenodo