

Bründl, Patrick et al.

Article — Published Version

Semantic part segmentation of spatial features via geometric deep learning for automated control cabinet assembly

Journal of Intelligent Manufacturing

Provided in Cooperation with:

Springer Nature

Suggested Citation: Bründl, Patrick et al. (2023) : Semantic part segmentation of spatial features via geometric deep learning for automated control cabinet assembly, Journal of Intelligent Manufacturing, ISSN 1572-8145, Springer US, New York, NY, pp. 1-15, <https://doi.org/10.1007/s10845-023-02267-1>

This Version is available at:

<https://hdl.handle.net/10419/312467>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.



<https://creativecommons.org/licenses/by/4.0/>



Semantic part segmentation of spatial features via geometric deep learning for automated control cabinet assembly

Patrick Bründl¹ · Benedikt Scheffler¹ · Micha Stoidner¹ · Huong Nguyen¹ · Andreas Baechler² · Ahmad Abrass² · Jörg Franke¹

Received: 25 February 2023 / Accepted: 4 November 2023
© The Author(s) 2023

Abstract

Industries with batch size one manufacturing philosophies for highly customized products have been largely limited in manufacturing automation. The control cabinet industry is particularly affected by this problem during the mounting and wiring of components due to high variety, variance, and complexity of components as well as handling tasks. Rapid advances in the field of machine learning are opening new possibilities for automating previously manual processes. This paper proposes a concept for identifying geometric features of electrical components that starts from STEP files and transforms them into modular metrics relevant to build a digital twin and (automatic) manufacturing. The architecture is tested on a self-aggregated and processed dataset of control cabinet components and achieves an average dice score of 65.27% and an intersection over union of 51.41% across all segmentation classes. In addition to semantic part segmentation of the components, the cluster, volume and surface centroids, the normal vectors and the size of each feature are computed. The paper evaluates the suitability of cutting-edge techniques such as diffusion as well as established deep learning architectures. The result is a hybrid end-to-end inference pipeline suitable for general spatial assembly processes.

Keywords Geometric deep learning · Control cabinet · Digital twin · Surface mesh processing · Semantic part segmentation

Introduction

Preparatory steps in control cabinet assembly, such as wire cutting, stripping, attaching, and crimping the ferrules, can be carried out by semi- or fully automatic machines (Stoidner et al., 2023). In contrast, the mounting of the components and their wiring is almost exclusively done manually by skilled workers. The wiring of the individual components is also the most time-consuming production step (Gausemeier et al., 2016) and takes up to 49% of the total assembly time (Spies et al., 2019). The high complexity of wiring can be broken down into separate steps such as reading and interpreting the circuit diagram, finding the source and destination, and connecting the wires to the terminal points.

The share of mechanical assembly consisting of mounting and wiring is about 75% of the processing time (Tempel et al., 2017). In contrast to other already semi- or fully-automated manufacturing steps, mechanical assembly benefits much more from the potential of digitization and automation using modern machine learning (Tempel et al., 2017). Control cabinet manufacturers see the future of control cabinet construction in digital manufacturing, where wiring is automated or digitally assisted (Tempel et al., 2017). So far, there are only very few approaches to this, which are also limited to the automation of series products (Spies et al., 2019). Control cabinets are often engineer-to-order products and are manufactured in small batch sizes with a large variety of components (Bründl et al., 2023). Due to this high level of complexity, there is neither a concept for the automatic wiring of these nor a procedure for analyzing the components to extract important information for generating a digital twin of a control cabinet (Großmann et al., 2017; Linsinger et al., 2018). Previous approaches only pursue the automated recognition of switches in switch cabinets based on RGB images (Zhang et al., 2020) as well as template matching

✉ Patrick Bründl
patrick.bruendl@faps.fau.de

¹ Institute for Factory Automation and Production Systems (FAPS), Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany

² Rittal GmbH & Co. KG, Herborn, Germany

algorithms (Hefner et al., 2021) for the identification and orientation determination of components in the control cabinet using images and point clouds. However, the origin of all approaches in Spies et al., 2019, Hefner et al., 2021 and Hefner et al., 2020 lies in data from actual manufacturing, instead of raw CAD models, which themselves possess more relevant information for automation tasks. However, only one ECAD manufacturer, in a diverse market, allows the input of assembly-relevant information in their data standard. Furthermore, this lone standard provides limited details, omitting crucial aspects like tool insertion, mounting rail attachment, or device labeling. Most component manufacturers fail to provide this data, leaving the universally available Standard for the Exchange of Product Model Data (STEP) file model as the primary source of information for engineering and manufacturing planning. The vast array of components and variants presents a challenge that traditional image processing struggles to meet. Even though machine learning could offer solutions, its adoption is used very little and methods used are not state-of-the-art. Moreover, from a machine learning viewpoint, many existing datasets, such as SHREC or ShapeNet, are constrained by their low-resolution meshes, making them unsuitable for many real-world use cases. Accordingly, the novelty lies in the combination of various already evaluated and partly established procedures. Since a CAD model is available for each component, this also represents the logical starting point for the assembly process. Since the assembly and wiring of components in control cabinet manufacturing is challenging due to the high variety, variance and complexity of the components as well as the handling tasks and a lack of reliable data, a general concept for the extraction of geometrically relevant information is needed.

For automated wiring, information about the type of source and target as well as their positions is required (Tempel et al., 2017). Hence the main task to cover both cases together is therefore the identification and positioning of the relevant features of a component, as shown in Fig. 1. For the installation of the components on mounting rails knowledge of the position of the snap-on points is therefore required. For the wiring of the attached terminal blocks, their size and orientation are required in addition to the position. For a fully automated assembly and wiring, knowledge about the position, size and normal vectors of the snap-on points (red vectors), tool and cable entries (green vectors), and ideally labeling areas (orange vectors) is required. This paper presents a methodology for capturing these features, which is also generally suitable for any spatial assembly and can therefore be used in further industries.

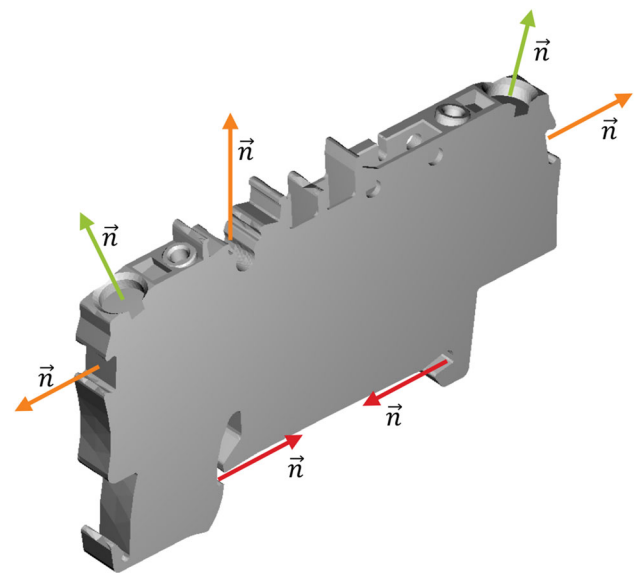


Fig. 1 Exemplary identification of the position, size, and normal vectors of the features of the control cabinet components

Related work

State-of-the-art research for 2D-based detection systems is only suitable for triangle meshes under very specific conditions (Bronstein et al., 2016). More successful methods fall into the area of geometric deep learning (GDL) which make use of intrinsic data operations on the meshes and segment the data intrinsically (Bronstein et al., 2016, 2021).

Intrinsic operations

The increasing interest in applying machine learning (ML) to geometric data has led to trying out principles on 3D geometric data that have been successful on 2D data (Bronstein et al., 2016) such as multi-view CNNs (Su et al., 2015). The main drawback of this approach is the treatment of non-Euclidean as Euclidean structures (Bronstein et al., 2016). This must be avoided for two reasons. First, the Euclidean representation of complex 3D objects, such as depth images or voxels, can cause essential parts of the object or fine details to be lost or even destroy the topological structure. Second, Euclidean representations are not intrinsic and change depending on the pose or deformation of the object (Bronstein et al., 2016). Symmetry is a profound concept in GDL since it includes all properties related to transformations, such as translation, reflection, rotation, scaling, and permutation. In the literature, invariance and equivariance are usually used in this context to express the behavior of a mathematical function in terms of the transformation (Atz et al., 2021). Achieving invariance to shape changes is a common requirement in data processing and requires very complex models as well as large training

sets due to a large number of degrees of freedom involved in describing non-rigid deformations (Bronstein et al., 2021).

3D Part segmentation

Current deep learning approaches achieve significantly better performance in 3D sub-segmentation of surfaces in comparison to traditional computer vision approaches (Rodrigues et al., 2018). The literature has proposed very different methods over the years starting with Shape projective Fully Convolutional Networks, which segment the surface based on multiple 2D projections (Kalogerakis et al., 2016). The Voxel Segmentation Network was the first method operating directly on 3D data (Wang & Lu, 2018). This was followed by PointNet (Qi et al., 2016), PointNet++ (Qi et al., 2017), and PointCNN (Y. Li et al., 2018), which implemented stepwise permutation invariance over the input order of points, recursive application of the network to local clustered points, and generalized convolutional layers (Schneider et al., 2020). The implementation 3D data segmentation utilizing such models in conjunction with synthetic data in the industrial environment for complex wiring system assemblies has shown high potential under the premise of a large and high-quality training data set (Nguyen et al., 2022). While these approaches work to some degree, they are not deformation invariant, which is why geometric neural networks have gained popularity, especially for part segmentation of triangle meshes (Schneider et al., 2020). Unlike these approaches, MeshCNN (Hanocka et al., 2018) and DiffusionNet (Sharp et al., 2020) operate directly on the data and fall into the scope of geometric neural networks.

Graph representation learning

The domain of GDL is divided into the 5Gs: Grids, Groups, Graphs, Geodesics and Gauges (Bronstein et al., 2021). While the first two Gs play no role in triangle meshes, graphs are a useful data structure for representing triangle meshes (Bronstein et al., 2016, 2021). Networks operating on this structure fall in the domain of Graph Representation Learning (Bronstein et al., 2016). A well-known network in this scope is the MeshCNN presented by Hanocka et al., 2018 which introduced an intrinsic convolution and pooling as well as unpooling operation on meshes. In combination with common architectures such as the U-Net by Ronneberger et al., 2015, triangular meshes can be classified or semantically segmented (Bronstein et al., 2021) on an edge (Hanocka et al., 2018), face or vertex basis (Sharp et al., 2020). Graphs, denoted by $G = (V, E)$ are usually represented for this purpose using vertices, $V = \{v_1, v_2, \dots, v_n\}$ and edges $E = \{e_{i,j}\}$ whereby $e_{i,j}$ connects vertex v_i to v_j (Chen et al., 2020). Here, the representation of all connections of

a graph is realized using an adjacency matrix. For computational reasons, a representation using a sparse matrix may also be appropriate (Schneider et al., 2020).

Learning on manifolds

As an alternative to learning features of triangle meshes using graphs, this is also possible with geodesics and gauges. In the context of classification by the 5Gs, meshes fall between graphs, geodesics, and gauges. While meshes are in a sense very similar to graphs, they can also be considered as continuous objects. For this purpose, the representation of graphs is extended over triangles consisting of three vertices each. The additional representation of a mesh consisting of vertices, edges, and faces $T = (V, E, F)$ can therefore be seen as an approximation of a surface depending on the triangle resolution. The approximation of a continuous surface is particularly useful because meshes can be described by topological invariants such as the Euler characteristic. Independently of the curvature in space, these do not change. For meshes, the Euler characteristic is described by $\chi = V - E + F$. For the convex manifolds used in this work, the Euler characteristic is $V - E + F = 2$ (Flegg, 2001; Lee, 2011). Very good results can be obtained on manifolds through the calculation of geodesics, parallel transport, and local gradient spatial features (Sharp et al., 2020).

Dataset

The dataset used in this work was independently aggregated, preprocessed, labeled, and augmented due to lacking open source and domain-specific dataset. The generated dataset is based on real control cabinet components. The dataset generating process steps are described in the following sections and the data used is made openly available (Scheffler & Bründl, 2023).

Data aggregation

Since most manufacturers of electrical components make their models freely available online, web scraping offers the best solution for aggregating the data. Therefore, use was made of the web browser automation library Selenium (SeleniumHQ) to scrape the CAD data. Proprietary file formats from ECAD software companies, which are often used in the modeling of control cabinets, are not suitable due to the lack of possibility to read the files without a license. The models aggregated for this work are therefore exclusively STEP files and include 46,068 STEP models from 75 different manufacturers. Since most STEP files cannot be matched to the metadata contained on the website after downloading due to a lack of information, the data aggregation was designed

sequentially so that one model was always aggregated followed by the associated metadata. Models were named after a unique manufacturer product number and stored in folders for each manufacturer separately. This prevented overwriting models during the data aggregation in case of identical product numbers of different manufacturers. The accumulated metadata was stored in a dictionary with mapping tables indexed by product number in a parent dictionary, allowing information retrieval of a component in constant time. Aggregated metadata include the following information: part number, manufacturer, part type, technology, category, subcategory, series, width, height, length, current, description, and the official documentation URL. Since many components differ only in their electrical characteristics but have identical geometry, all files were hashed using a Secure Hash Algorithm 256 to identify geometric duplicates. Filtering of the entire dataset took place based on the category, subcategory, and technology.

Data preprocessing

During data pre-processing, a STEP file is converted to a triangle mesh, as these are much easier to represent in tensors (Zhou et al., 2018). Successful approaches of semantic segmentation of triangle meshes as described in Schneider et al. (2020), Hanocka et al. (2018) or Sharp et al. (2020) are currently based on OBJ or OFF files, which is why the STEP files are converted into these formats. Since the number of vertices per mesh varies greatly after the conversion all files are scaled uniformly to 6000 vertices and possible errors are corrected so that final 2-manifolds are available. Scaling and error correction were performed using the ManifoldPlus algorithm proposed by (J. Huang et al., 2020) and based on (J. Huang et al., 2018), in which the meshes are represented as octrees and the surface is constructed by iso-surface extraction (Visual Computing for Medicine, 2014). In addition, the STEP models are rendered from six views, specifically the top, bottom, front, back, left, and right to create grayscale images of 512×512 pixels, as shown in Fig. 2. The images are also labeled in the same four classes using LabelStudio, omitting the housing as the background class. The finished labels were then exported in COCO-JSON format (Tkachenko et al., 2020–2022).

Data preparation

To train a neural network for the semantic segmentation of triangle meshes, three different labeling possibilities can be considered. These are labeling of the edges (Hanocka et al., 2018), the vertices or the faces (Sharp et al., 2020). For this paper, the nodes of the meshes were labeled and converted to edge labels for alternative network architectures. The labeling was implemented using Blender v3.3.0, while

the extraction of the labeled data from Blender was automated using a script. For this purpose, the library bpy was used to export the labels as vertex groups from Blender and to convert them into a tensor representation. This is scalable for large datasets, contrary to previous approaches as described in (Schneider et al., 2020). The triangle meshes were divided into 5 classes, some of which are shown in Fig. 1 and all of them in 3:

- Housing (dark blue)
- Contacting (green)
- Snap-on point (red)
- Cable entry (yellow)
- Labeling area (orange)

After preprocessing, eliminating duplicates, and filtering for relevant files, a total of 234 files were labeled. The labels were stored as row tensors in text files.

Data augmentation

To increase the size of the dataset and to achieve the goal of neural networks with high generalization capability, the data was augmented (S.-G. Huang et al., 2021). Random rotations, noise, and deformations were added. Noise was implemented by multiplying the vertices by a value close to 1. The multipliers were sampled from a log-normal distribution with a mean μ of 0 and a standard deviation σ of 0.005. Generating a tensor of identical size to the vertex tensor and then element-wise multiplying both tensors result in a triangle mesh with noise based on this log-normal distribution. After adding the noise, it is smoothed element-wise using a Laplace filter (Visual Computing for Medicine, 2014) (1) as implemented in the Open3D library (Qian-Yi Zhou et al., 2018).

$$v_i' = v_i \cdot \lambda \sum_{n \in N} w_n v_n - v_i \quad (1)$$

Another possibility for the augmentation of the triangle meshes is the deliberate deformation based on the As-Rigid-As-Possible algorithm (Sorkine & Alexa, 2007). In this work, between one and ten vertices were randomly selected in the four geometric feature groups of contacting, snap-on point, cable entry, and labeling area. From the selected vertices, the Frobenius- or squared L^2 -Norm to all vertices of the mesh and the 50% percentile of the norm are calculated according to (Iserles, 1990). All vectors whose distance is above the 50% percentile are assigned to the static vertices and are not deformed. Thus, deformations only affect the local structure of the mesh near the randomly selected vertices. Deformation of the dynamic vertices is performed for all selected vectors until the desired number of deformations is reached. Figure 3

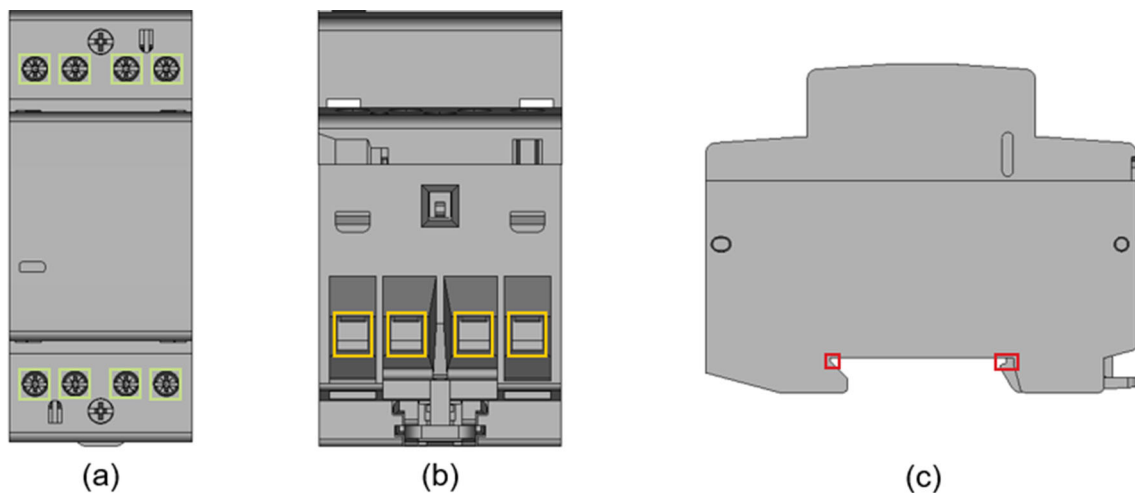


Fig. 2 Labeled grayscale images from top (a), front (b), and left (c) with green marking for contactings, red denoting snap-on points, and yellow corresponding to cable entries

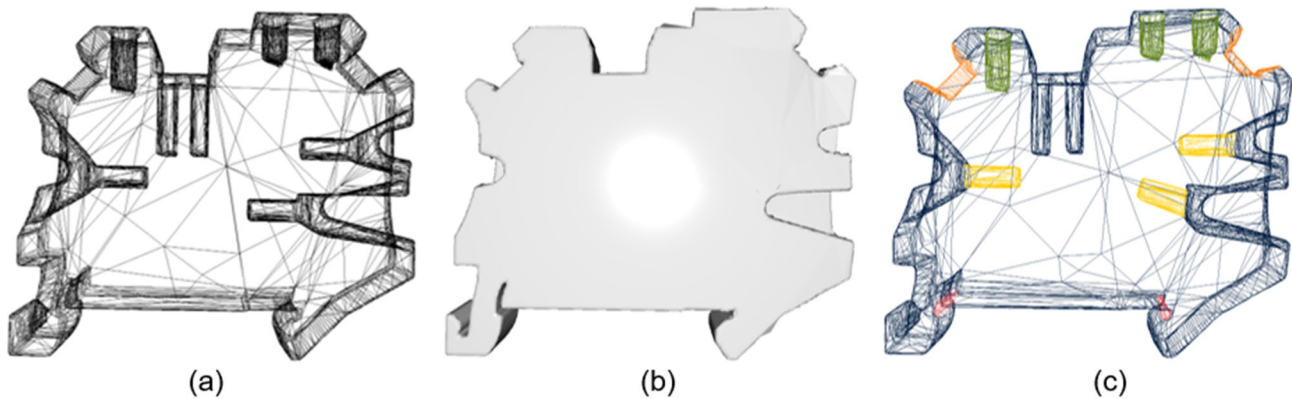


Fig. 3 Comparison between normal series clamp (a), with log-normal noise (b), and smoothed via Laplace filter as well as deformed by As-Rigid-As-Possible algorithm (c)

depicts portions of the data augmentation, where (a) represents the normal terminal block, in (b) noise has been added, and in (c) the spikes created by the noise were smoothed via a Laplace filter and multiple As-Rigid-As-Possible deformations were carried out. In each case, the deformations are only considered as valid if the topological invariants of the 2-manifold did not change. Deformations resulting in self-intersecting surfaces are therefore avoided. Overall, the training and validation dataset was doubled in size by the augmentation. The test data set remains unchanged.

End-to-end inference of component features

Different network architectures were implemented and investigated, which partly complement each other or represent alternatives to each other. Two geometric neural networks and a current version of the established YOLO series were

implemented in PyTorch (Paszke et al., 2019). The DiffusionNet was additionally evaluated as an alternative to the MeshCNN, since the latter has a tendency to overfit on the triangle mesh structure instead of learning the relevant features (Sharp et al., 2020). Since the number of labels is highly variable and uneven, a classical 2-dimensional approach was additionally considered to counteract this imbalance. Moreover, it turns out that for a class like snap-on points, the 2-dimensional determination of the position of visible features and thus, the bounding box is not only sufficient but also computationally less expensive. The predictions are subdivided into individual instances, which are used to calculate various metrics relevant to the assembly and wiring of control cabinets.

Inference pipeline

The entire inference pipeline is built from the described steps in “Dataset” section, the modeled neural networks and the

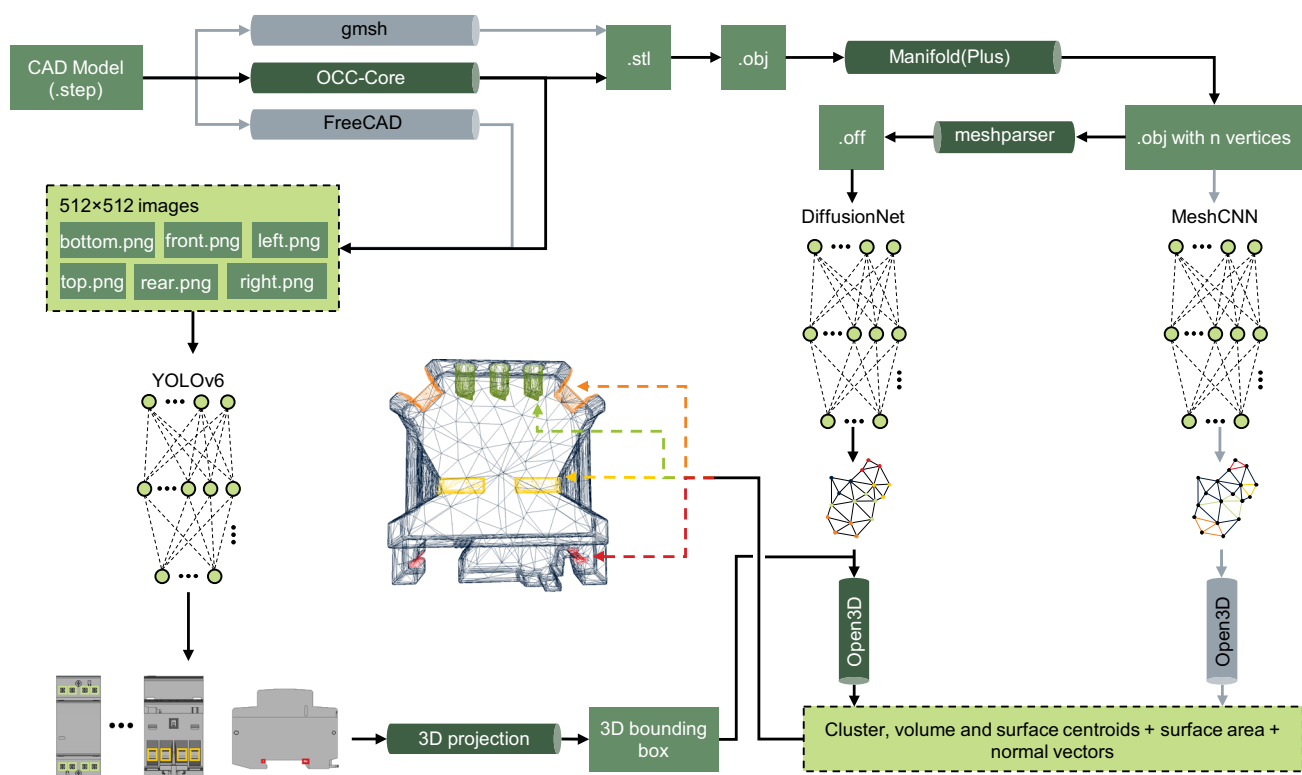


Fig. 4 End-to-end inference of the presented procedure

calculation of the final metrics of “**End-to-end inference of component features**” section. Figure 4 illustrates the data flow as well as the interrelationships of the individual steps. First, the STEP file is converted to an STL file using the Python OpenCascade Core CAD interface (tpaviot, 2023). Simultaneously, six gray-level images of the STEP file are generated. While the images are inferred directly via the YOLOv6 CNN, the STL file is first converted into an OBJ file to correct possible errors during conversion using ManifoldPlus and then scaled to 6,000 vertices. This file can be used as input for the MeshCNN and is converted into an OFF file to alternatively be inferred via the DiffusionNet. For performance reasons, the conversions to OFF files were implemented via the Python C API itself. Since the results of the YOLO network are only two-dimensional, they are projected onto the triangle meshes and the bounding box is extended by the missing dimension. The added side of the bounding box has the entire width of the component in that dimension. Subsequently, the vertices within the bounding box are cropped out, resulting in an identical situation to the output of the two geometric meshes. Based on these segmentation masks, the centroids, size and normal vectors are calculated.

MeshCNN

The MeshCNN implemented in this work is based on the mesh convolution, mesh pooling, and mesh unpooling operation proposed by Hanocka et al., 2018. Convolution is performed on each of the four adjacent edges as a dot-product of a kernel k and the adjacent edges. Accordingly, the convolution of an edge feature e and the neighboring edges is

$$e \cdot k_0 + \sum_{j=1}^4 k_j \cdot e_j \quad (2)$$

where e_j is the feature of the j th convolution neighbor of e (Hanocka et al., 2018). Due to the irregular structure of graphs and meshes as opposed to images, the pooling operation is performed as a series of edge collapses. Five edges are combined to form two edges. The edge collapse is performed until a target number of edges is reached, which results in temporary simplified representations of the graph. The prioritization of the edges to be collapsed is done by the norm i.e., the length of the edges under the assumption that shorter edges encode less information. While convolution and pooling are used in the U-Net’s encoder, they are partially reversed using cached information from the original mesh through a mesh unpooling operation. (Hanocka et al., 2018).

DiffusionNet

The diffusion net was used with a weighted Negative Logarithmic Likelihood Loss. The weights counteract the distribution of the labels of the training data. The diffusion was calculated using spectral acceleration instead of a direct implicit Euler time step. The eigenbasis was precomputed and cached allowing the diffusion to be evaluated directly at each time t by element-wise exponentiation. As Sharp et al. (2020) suggests, the approximation error has a negligible effect on the result. The Laplacian L and the mass matrix M are the eigenvectors $\phi_i \in \mathbb{R}^V$ solutions for (Sharp et al., 2020):

$$L\phi_i = \lambda_i M\phi_i \quad (3)$$

These correspond to the first k smallest-magnitude eigenvalues $\lambda_1, \dots, \lambda_k$. The diffusion of u at timestamp t is then evaluated by projecting onto the spectral basis, evaluating pointwise diffusion, and projecting back

$$h_t(u) := \Phi \begin{bmatrix} e^{-\lambda_0 t} \\ e^{-\lambda_1 t} \\ \dots \end{bmatrix} \odot (\Phi^T M u) \quad (4)$$

where \odot marks the Hadamard product. Since no non-rigid invariance is required, only the 3D coordinates are used as input features instead of the heat kernel signature (Sharp et al., 2020).

YOLO model

A YOLO network was implemented to identify the number of instances per feature. The YOLO series is the most established detection system in industrial applications due to the balance between speed and accuracy. While the YOLO framework describes a novel approach for single-level detectors in the first three versions, the framework was broken down into separate components in version four to integrate different network backbones (Redmon et al., 2015). The most successful version of the YOLO series at the time of this work is YOLOv6, which was implemented as part of this work according to C. Li et al., 2022. A single-stage object detector consists of three parts, backbone, neck, and head. The backbone mainly determines the feature representation capability. Having a high proportion of the total computational load, the backbone has a significant impact on the inference time (Bochkovski et al., 2020). The second part is used to combine low-level physical features with higher-level semantic features. The third part includes several convolutional layers and is used to predict the results based on features of the previously traversed levels. YOLOv6 is built on the Reparametrization Visual Geometry Group

(RepVGG) architecture (Ding et al., 2021) which itself is based on a Path Aggregation Network (Liu et al., 2018). Since object detection involves both classification and localization, two loss functions are needed. The efficiently decoupled head at the end of the network from YOLOv5 separates the network into classification and regression loss functions. (Liu et al., 2018) The VariFocal loss function is used for classification and a variant of the Intersection over Union (IoU) loss function is used for regression (C. Li et al., 2022). The choice of this network architecture is based in particular on the underlying RepVGG architecture, which is kept simple, fast and memory efficient (Ding et al., 2021).

Instantiation based on graph connectivity

Since the output of the neural networks only corresponds to semantic segmentation but not to instance segmentation, a subsequent subdivision of the predictions into individual instances is necessary. By filtering the predictions by each category, each resulting subgraph forms an instance. To filter individual incorrectly segmented clusters, a minimum number per class is set as a threshold to be recognized as correctly classified. Otherwise, these vertices are assigned to the background class. Since additional problems can arise in the separation of features that are very close to each other, the outermost layers of the graphs of the individual features were removed. The elimination of the labels is based on the neighboring labels. All vertices which are not exclusively surrounded by identical labels were assigned to the housing.

Feature metrics

The information content of the pure predictions of the neural networks is very high, but without post-processing, it is of little importance for the assembly and wiring of the control cabinet components. Thus, the calculation of fitting metrics is required and was implemented via Open3D (Qian-Yi Zhou et al., 2018) and the scientific array library NumPy (Harris et al., 2020). The metrics include the position consisting of x, y and z coordinates, the normal vectors, and the size of the feature instance as shown in Fig. 5. The position is represented in triplicate using different centroids. First, the cluster centroid was calculated, which represents the center of the vertices of the segmented triangle mesh of a feature. The position is calculated in triples as the cluster centroid, volume centroid, and surface centroid of the vertices. Unlike the trivial cluster centroid, the volume centroid cannot be calculated directly since surface triangle meshes do not have a volume. However, for highly curved surface meshes, this metric is still feasible. Thus, the convex hull is computed before its volume centroid. The surface centroid is most interesting since it lies directly in the vector space of the 2-manifold. Lastly, the centroid of area is of particular interest for contacting and cable

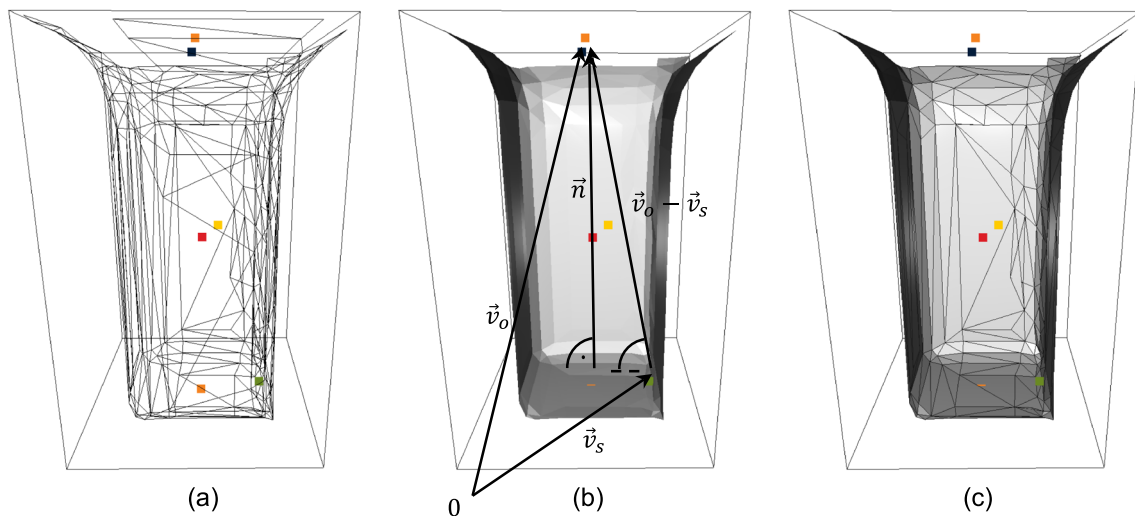


Fig. 5 Cluster (yellow), volume (red), surface (green) centroid, as well as normal vector (dark blue) of a cable entry segmented by the DiffusionNet

entry applications as it can be used to determine the depth of the cavity.

The centroid of the surface is determined by iteratively eliminating the outermost edge vertices. This is repeated until either only edge vertices are left or the triangle mesh consists of only one point, which represents the centroid of the area. On closer inspection, it becomes obvious that this is not the centroid of the manifold but from the underlying mesh structure. Therefore, before applying this method, the triangle mesh is uniformly subdivided via smooth subdivision, which minimizes this effect (Loop, 1987). Nevertheless, it is evident from Fig. 5a–c that this point may still deviate from the actual surface centroid even after smooth subdivision.

In addition to the surface centroid, the center of all edge vertices is calculated last as shown in Fig. 6, which is close to the center of the cavity (dark blue) of a cable entry or contact. Optimally, this is opposite the center of gravity of the surface and in the center of the geometry's opening. For a more exact calculation, however, this new vector is subtracted from the surface centroid as in Fig. 5b. The result then provides information about the orientation of the geometry.

From the difference of the vector v_o to the center of the edge vertices and the vector of the surface centroid v_s , the normal vector of the feature can be determined. Due to the approximate determination of the centroid of the surface, this normal vector may also be inaccurate. To align the normal vector in the correct direction perpendicular to the mean surface, the vector is aligned parallel to two sides of the bounding box of the feature. This automatically results in an orthogonal vector \vec{n} at the center of the feature. The correction and alignment of the normal vector takes place in the two directions whose angles to the vector are minimal. After the alignment adjustment, the vector is placed in the center of the bounding box, ultimately pointing from the bottom to the top

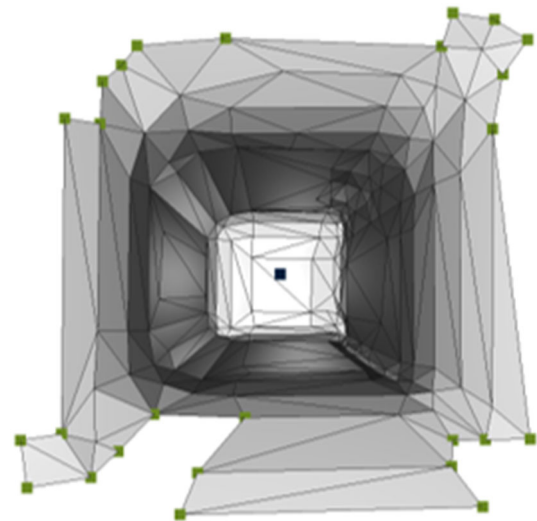


Fig. 6 Approximation of the vector of a geometric feature using the example of a cable entry point

orange point. Finally, the size of each feature is determined by calculating the sum of the surfaces of all triangles.

Results

All results were achieved under identical conditions. The graphics card used was an NVIDIA GeForce RTX 2080 Ti with 11 GB GPU memory and the hardware accelerator was CUDA Toolkit 11.4 with driver version 470.57.02. The PC had 32 GB RAM and an Intel Core i7-6700 K with 4 physical CPU cores. The entire inference runs on Ubuntu 20.04.03 LTS, since libraries, which currently have Linux-only support, were tested during development. However, the

Table 1 Weighting of the individual segmentation classes according to their distribution in the training dataset

Class	Weight	Percentage of labels
Housing	46.4605	70.7
Contacting	221.5453	14.32
Snap-on point	1156.4556	2.83
Cable entry	546.8523	5.74
Labeling area	514.3766	6.41

presented pipeline can also be used on other operating systems. To avoid errors in the calculation of the neural network metrics, they were calculated using the torchmetrics library (Nicki Skafte Detlefsen et al., 2022). In contrast to Hanocka et al., 2018 and Sharp et al., 2020 the predictions were not evaluated based on accuracy but on the Jaccard index like proposed in Schneider et al., 2020 and Dice score. These are more informative, especially for highly imbalanced datasets in segmentation tasks (Schneider et al., 2020). The dataset was split for all models in the same 70/20/10 train/val/test-split. Random data splitting was used to avoid introducing human bias.

Hyperparameter optimization

The hyperparameters of DiffusionNet were optimized with Ray Tune using an Asynchronous Successive Halving Algorithm (ASHA) (Liaw et al., 2018). Using this scheduling algorithm under high parallelism, which performs aggressive early stopping, a sufficiently large parameter space could be excluded to determine the final parameters manually via random search (Feurer & Hutter, 2019; L. Li et al., 2018). A weighted loss function is recommended based on the very high imbalance of the individual classes of the labels. The loss function of MeshCNN and the DiffusionNet were weighted according to Table 1. These approximate the inverse fraction of the percentage of classes of all labels.

The batch size of the two geometric networks were set to only one sample owing to the limitation of the graphics memory of 11 GB. The hyperparameters for training the diffusion net amount to the values in Table 2, with the studied intervals on the left and the optimized values on the right.

Since MeshCNN requires almost 20 times longer training time compared to the DiffusionNet, the hyperparameters for the former were only optimized manually based on already existing results from previous studies (Schneider et al., 2020). The non-default options being used during training result in Table 3. A lambda learning rate policy means an individually implemented policy for learning rate decay. In this case the learning rate started decaying beginning with epoch 400

Table 2 Observed parameter space during hyperparameter optimization via Ray Tune using the ASHA scheduler as well as the optimized results per parameter

Hyperparameter	Sampling type	Sampled range	Optimized value
Input feature	Categorical	xyz, hks	xyz
Learning rate	Logarithmical	$[1e^{-5}, 1e^{-1}]$	$1e^{-3}$
Decay frequency	Categorical	25, 50, 100, 250, 500	100
Decay rate	Categorical	(0, 1)	0.75
Loss function	Categorical	NLL, CE	NLL
# diffusion blocks	Categorical	1, 2, 3, 4, 5	3
block dimension	Categorical	32, 64, 128, 256, 512, 1024	64
# eigenvectors	Categorical	32, 64, 128, 256, 512, 1024	64
Dropout	Uniform	(0, 1)	0.3
Output	Categorical	Vertices, faces	Vertices
Batch size	Categorical	1, 2	1

“#” the number of the affected parameter

Table 3 Hyperparameters for the MeshCNN deviating from (Hanocka et al., 2018) al following (Schneider et al., 2020)

Hyperparameter	Used value
Convolution filters	[32, 64, 128, 256]
Pooling resolutions	[9000, 4000, 2500]
Inputs edges	18,000
Residual blocks	3
Momentum term	0.9
Learning rate	0.002
Learning rate policy	Lambda
Mesh augmentations	1

slowly approaching zero. For the implemented YOLOv6-CNN, the hyperparameters recommended and validated from C. Li et al. (2022) were used.

DiffusionNet

Dice score and Jaccard index are more suitable in contrast to accuracy, which is rather uninformative in segmentation tasks. The DiffusionNet was trained over 800 epochs and achieved the best Jaccard index on the validation dataset in epoch 736. The associated model was used for the final metrics on the test dataset. The individual Dice scores and Jaccard

Table 4 Dice scores and Jaccard indices for the five segmentation classes housing, contacting, snap-on point, cable entry and labeling area predicted by DiffusionNet

Class	Dice-score	Jaccard-index
Housing	0.8576	0.7438
Contacting	0.6939	0.5925
Snap-on point	0.5261	0.3684
Cable entry	0.6686	0.4716
Labeling area	0.6055	0.3943

Table 5 Dice scores and Jaccard indices for the five segmentation classes housing, contacting, snap-on point, cable entry and labeling area predicted by MeshCNN

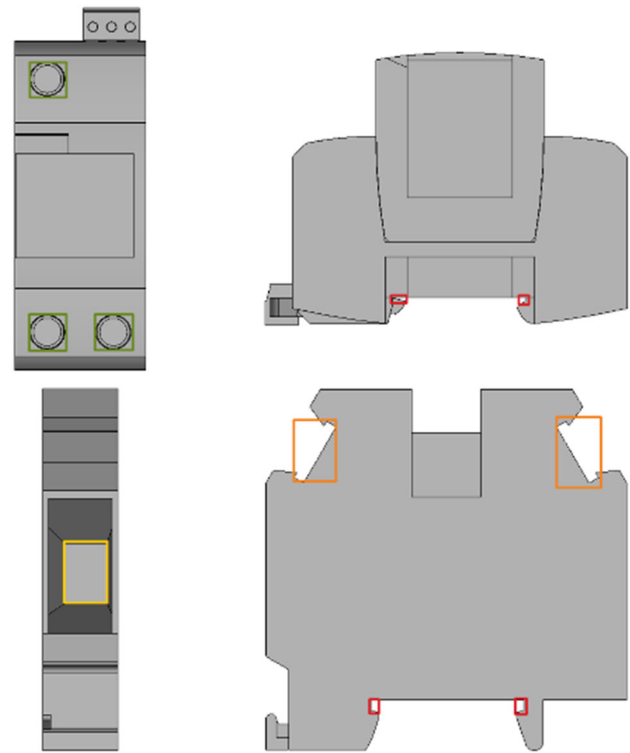
Class	Dice-score	Jaccard-index
Housing	0.8598	0.7597
Contacting	0.7561	0.6436
Snap-on point	0.4289	0.3062
Cable entry	0.4652	0.3739
Labeling area	0.4247	0.3461

indices for the classes correspond to the values in Table 4, resulting in an average Dice score of 0.6527 as well as an average Jaccard index of 0.5141 and a weighted Jaccard index of 0.7052.

The housing is by far the best segmented which can be explained by the frequency of the labels. This is partially compensated by the weighted NLL loss but cannot completely compensate this effect because of the ignorance of the distribution of the validation and test dataset. The contacts are segmented second best followed by the cable entries, labeling areas, and lastly the snap-on points. This order accurately reflects the distributions of the dataset with respect its data imbalance.

MeshCNN

The MeshCNN was trained to only 400 epochs as the computation takes ten times as long. The reason for this is the sequential execution of the mesh pooling and unpooling operation. Despite many measures such as L2 regularization, dropout, and smaller model sizes to reduce the trainable parameters, MeshCNN did not generalize well. This is reflected in diverging Jaccard indices and Dice scores, as more dominant labels such as housing are segmented significantly better than underrepresented labels such as snap-on points. The overall average Jaccard index is 0.4859 with a weighted Jaccard index of 0.7028. The individual metrics per class are listed in Table 5.

**Fig. 7** Predictions by YOLOv6 on two control cabinet components from two different views each for the detection of contactings (green), snap-on points (red), cable entries (yellow), and labeling areas (orange)

YOLOv6

The YOLOv6-CNN was trained for a total of 400 epochs and evaluated using the mean average precision. For the evaluation against the test dataset, the cached model from epoch 365 was used and achieves a minimum loss of 1.3697, which is composed of the four individual validation losses IoU loss (0.5731), L1 loss (0.1824), object loss (0.2802) and cross entropy loss (0.3392). The mAP@0.5 on the test dataset is 0.9311 while the mAP@[0.5:0.95] is 0.7279. The example inferences of the YOLOv6-CNN from Fig. 7 illustrate some results of the network.

Comparison

Figure 8 compares the predictions of two components of ground truth labels and the predictions inferred by MeshCNN and DiffusionNet. While both networks segment the component well, the DiffusionNet predicts with higher confidence. This is reflected in the many small false predictions of the MeshCNN, which are not present in the predictions of DiffusionNet. Latter tends to segment more rather than too little as evident in the segmentation of the snap-on points and labeling areas. Although this decreases the error metrics, it has a negligible effect on the final centroids and normal vectors.

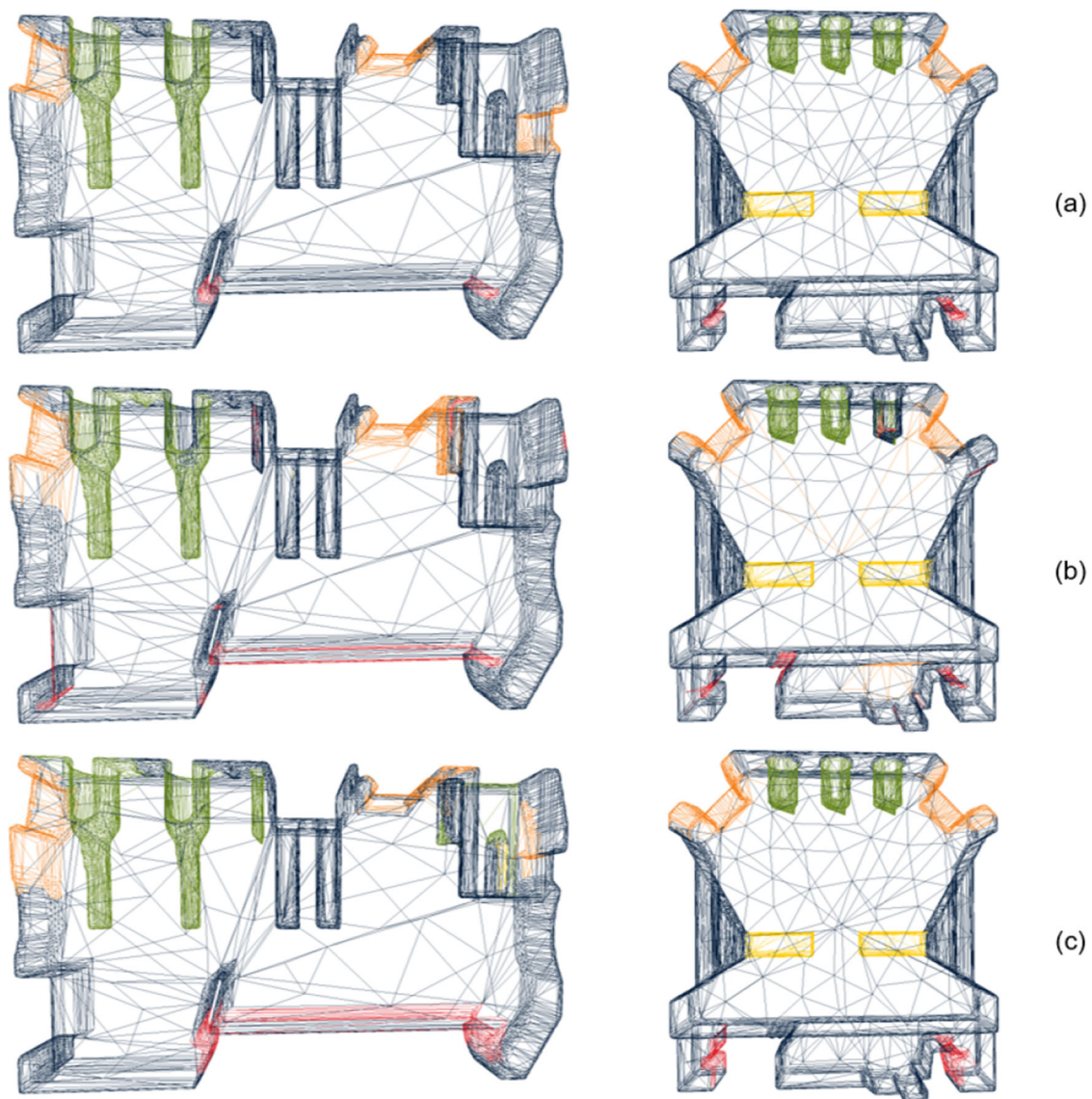


Fig. 8 Comparison of ground truth labels (a), the edge predictions of MeshCNN (b), and vertex predictions by diffusionNet converted to edges (c)

However, overconfidence can lead to erroneously connected clusters. Thus it can have a negative impact on nearby clusters of the same class.

Discussion and conclusion

The implemented YOLO network achieves very good results in all labeled classes. However, not all classes can always be detected from the six positions considered. This uncertainty about non-visible features from any of the six views therefore inevitably requires the additional use of another approach. Thus a hybrid inference is reasonable and YOLOv6 is therefore only used to infer the snap-on points in the final pipeline. Contact points that need to be contacted from an angle and

are partially hidden from the top or the side are an exemplary problem. Thus, failure to detect the contacting becomes much more likely. There is also the possibility that the same contacts are detected in different views and subsequent assignment is required or a connection is concealed from all six perspectives. The inference pipeline therefore ultimately uses the YOLO network to detect the snap-on points, since the visibility from at least two views is always guaranteed. Otherwise, clamping the component on a rail would not be possible. The use of both methods for different classes is therefore complementary, especially in the context that the DiffusionNet segments snap-on points worst due to the small number of vertices. The problem of poor results due to fewer labels can thus be effectively counteracted.

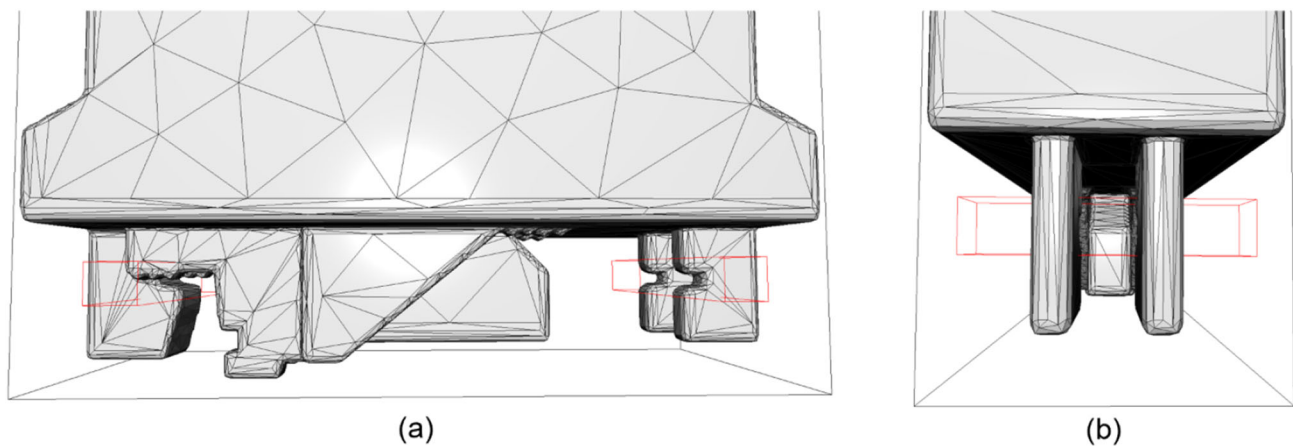


Fig. 9 Detection of the snap-on points by two-dimensional YOLOv6 predictions mapped to the triangle meshes

Figure 9 shows that the detection of the snap-on points by the YOLO network helps with another problem. In (a) it can be seen where the two rails run and where the component is mounted. In (b) the snap-on points of the right rail in (a) are divided into two parts. By segmenting via the DiffusionNet, the assignment of all segmented vertices of the class snap-on point to the correct rail in each case is very difficult in such cases, which is why the approach using the YOLO-CNN is helpful here.

The modeled MeshCNN can successfully segment the triangular meshes of the components. However it requires more than 20 times the duration during training compared to DiffusionNet. Thus, the hyperparameters can be optimized much worse since the many individual training runs are also more time-consuming. The output is a tensor with 18,000 entries representing the edge features. The posterior calculation of the metrics, which rely in part on the vertices to compute cluster, volume, and surface centroids, requires a mapping of edge to vertex features. Additionally, despite similar segmentation values, there is a much higher variance within the five classes. The weighted loss function did not have a sufficient effect to compensate for the widely varying frequencies of the labels. In addition, there is a stronger tendency to overfitting. This also confirms the claim made by Sharp et al., 2020 of a high tendency to overfit the MeshCNN to the mesh structure without learning the important geometric features of the manifold. This poses a crucial role, especially when using different mesh algorithms and mesh subdivisions. In contrast, the DiffusionNet could be trained in much shorter time and lower variance within the segmentation classes. A complete elimination of the dominance of the housing labels could not be counteracted here either.

However, the effect is smaller and all metrics per class are closer together. Unlike MeshCNN, DiffusionNet infers vertices directly. Thus, a subsequent conversion of predictions

is needed. Training and inference over faces are also possible. Likewise, with DiffusionNet, adaptation is possible to process point clouds (Sharp et al., 2020). Even though both options were not used in the context of this work, DiffusionNet offers a more flexible model architecture. A disadvantage of the DiffusionNet is the calculation of the Laplace, mass, gradient matrix, as well as several eigenvectors. In the training run, this is negligible because the matrices are only calculated once and then cached. However, during inference on real data, these matrices must be recalculated for each component. This results in an approximate inference time of the modeled DiffusionNet of about one second per component. The MeshCNN does not exhibit this problem but is very memory inefficient and leads to larger fluctuations in GPU utilization caused by very large adjacency matrices. As both networks have a high inference time, there is a potential for optimization in further research. Since both models segment the snap-on points of the components worst, it is obvious to include another inference alternative. The YOLO-CNN is therefore used exclusively for the snap-on points, since only with this class the visibility of the features of two of the six views is guaranteed. Complementarily, for all snap-on points the feature is always present across the entire width of the component, which is why it is not only easy to detect from a two-dimensional perspective, but the predicted bounding box can also be directly transferred into three dimensions, knowing the perspective.

The conclusive evidence presented in this study underscores the robustness and efficacy of an integrated approach combining geometric and conventional CNN for semantic part segmentation of spatial features. This method proves to be the method of choice, especially in light of the strengths of DiffusionNet, solidifying it as the recommended model. Yet, as illustrated in Fig. 8, partitioning the segmentation into individual instances remains a formidable challenge. Even though direct instance segmentation is well-established in

image processing, its extension to triangle meshes is currently under exploration. Therefore, future research should consider an end-to-end approach in line with architectures such as Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al., 2015), or Mask R-CNN (He et al., 2017) for direct instance inference.

This study underscores the robustness and broad applicability of a novel approach that adeptly generates relevant position data for automated wiring and assembly. By freeing the process from reliance on proprietary component formats and ECAD manufacturers, it offers the potential to revolutionize practices in industries often grappling with poor data quality and a prevalent lot size of one, such as control cabinet manufacturing. The approach proves particularly beneficial by significantly reducing manual data maintenance effort. With only a fraction of electrical and electronic components being fully maintained according to current data standards and the quality and completeness of the data often being inconsistent, this reduction represents a substantial milestone and a catalyst for industry-wide digitization and automation solutions.

Despite its evident benefits, the subdivision of segmentation into individual instances, especially for adjacent wiring points, presents a need for further research. It is thus crucial that future studies focus on enhancing the efficacy of this process and evaluating the absolute accuracy of the resulting positions. Such endeavors will confirm the approach's suitability for worker assistance systems or automation solutions, consequently opening avenues for more streamlined practices.

The approach's adaptability to other processes is significant, making it particularly relevant for other assembly procedures that depend on force-fit and material-locking connection technologies. These findings, along with the potential for further optimization and the clarity of future research directions, solidify the broad academic and industrial relevance of this study.

Acknowledgements The research presented in this paper was funded by Rittal GmbH & Co. KG in the course of the project DigiSchalt (Digital Transformation of the Control Cabinet Industry).

Funding Open Access funding enabled and organized by Projekt DEAL. Rittal GmbH & Co. KG funded a position at Institute FAPS and provided equipment that was used for this study. Author Micha Herbert is also employed at Rittal GmbH & Co. KG.

Data availability The data that support the findings of this study are openly available in Electrical and Electronic Components Dataset in Harvard Dataverse at <https://doi.org/https://doi.org/10.7910/DVN/D3ODGT> (Scheffler & Bründl, 2023).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes

were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Atz, K., Grisoni, F., & Schneider, G. (2021). Geometric deep learning on molecular representations. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2107.12375>
- Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2004.10934>
- Bronstein, M. M., Bruna, J., Cohen, T., & Velicković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2104.13478>
- Bronstein, M. M., Bruna, J., Lecun, Y., Szlam, A., & Vandergheynst, P. (2016). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*. <https://doi.org/10.48550/arXiv.1611.08097>
- Bründl, P., Stoidner, M., Nguyen, H. G., Baechler, A., & Franke, J. (2023). Challenges and opportunities of software-based production planning and control for engineer-to-order manufacturing. In E. Alfnes, A. Romsdal, J. O. Strandhagen, G. von Cieminski, & D. Romero (Eds.), *IFIP advances in information and communication technology. Advances in Production management systems. production management systems for responsible manufacturing, service, and logistics futures* (Vol. 691, pp. 67–79). Springer. https://doi.org/10.1007/978-3-031-43670-3_5
- Chen, F., Wang, Y.-C., Wang, B., & Kuo, C.-C.J. (2020). Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing*. <https://doi.org/10.1017/ATSIP.2020.13>
- Detlefsen, N. S., Borovec, J., Schock, J., Harsh, A., Koker, T., Di Liello, L., Stancl, D., Quan, C., Grechkin, M., & Falcon, W. (2022). TorchMetrics—measuring reproducibility in PyTorch. <https://github.com/Lightning-AI/metrics>
- Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., & Sun, J. (2021). RepVGG: Making VGG-style ConvNets great again. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2101.03697>
- Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *The Springer series on challenges in machine learning. Automated machine learning* (pp. 3–33). Springer. https://doi.org/10.1007/978-3-030-05318-5_1
- Flegg, H. G. (2001). *From geometry to topology* (1st Dover ed; Reprint). Dover Publications. <http://www.loc.gov/catdir/description/dover031/2001032305.html>
- Gausemeier, J., Echterfeld, J., & Amshoff, B. (2016). Strategische Produkt- und Prozessplanung. In U. Lindemann (Ed.), *Handbuch Produktentwicklung* (pp. 9–36). Carl Hanser Verlag.
- Girshick, R. (2015). Fast R-CNN. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1504.08083>
- Großmann, C., Graeser, O., & Schreiber, A. (2017). ClipX: Auf dem Weg zur Industrialisierung des Schaltschrankbaus. In B. Vogel-Heuser, T. Bauernhansl, & M. ten Hompel (Eds.), *Handbuch Industrie 4.0 Bd.2* (pp. 169–187). Springer. https://doi.org/10.1007/978-3-662-53248-5_58
- Hanocka, R., Hertz, A., Fish, N., Giryres, R., Fleishman, S., & Cohen-Or, D. (2018). MeshCNN: A network with an edge. *ACM Transactions on Graphics*. <https://doi.org/10.48550/arXiv.1809.05910>

- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Del Fernández Río, J., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585, 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1703.06870>
- Hefner, F., Schmidbauer, S., & Franke, J. (2020). Pose error correction of a robot end-effector using a 3D visual sensor for control cabinet wiring. *Procedia CIRP*, 93, 1133–1138. <https://doi.org/10.1016/j.procir.2020.04.088>
- Hefner, F., Schmidbauer, S., & Franke, J. (2021). Vision-based adjusting of a digital model to real-world conditions for wire insertion tasks. *Procedia CIRP*, 97, 342–347. <https://doi.org/10.1016/j.procir.2020.05.248>
- Huang, J., Su, H., & Guibas, L. (2018). Robust watertight manifold surface generation method for ShapeNet models. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1802.01698>
- Huang, J., Zhou, Y., & Guibas, L. (2020). ManifoldPlus: A robust and scalable watertight manifold surface generation method for triangle soups. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2005.11621>
- Huang, S.-G., Chung, M. K., & Qiu, A. (2021). Fast mesh data augmentation via Chebyshev polynomial of spectral filtering. *Neural Networks: The Official Journal of the International Neural Network Society*, 143, 198–208. <https://doi.org/10.1016/j.neunet.2021.05.025>
- Iserles, A. (1990). Matrix computations (2nd edition), by G. H. Golub and C. F. Van Loan. Pp 642. £38. 1989. ISBN 0-8018-3772-3 (John Hopkins Press). *The Mathematical Gazette*, 74(469), 322–324. <https://doi.org/10.2307/3619868>
- Kalogerakis, E., Averkiou, M., Maji, S., & Chaudhuri, S. (2016). 3D shape segmentation with projective convolutional networks. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1612.02808>
- Lee, J. M. (2011). *Introduction to topological manifolds* (Vol. 202, 2nd ed.). Graduate texts in mathematics. Springer. <https://doi.org/10.1007/978-1-4419-7940-7>
- Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., & Wei, X. (2022). YOLOv6: A single-stage object detection framework for industrial applications. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2209.02976>
- Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., & Talwalkar, A. (2018). A system for massively parallel hyperparameter tuning. In *Proceedings of machine. learning and systems*. <https://doi.org/10.48550/arXiv.1810.05934>
- Li, Y., Bu, R., Sun, M., Wu, W., Di, X., & Chen, B. (2018). PointCNN: Convolution on \mathcal{X} -transformed points. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. <https://doi.org/10.48550/arXiv.1801.07791>
- Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1807.05118>
- Linsinger, M., Kutschinski, J., Stecken, J., & Kühlenkötter, B. (2018). Mensch–Roboter–Kollaboration im Schaltschrankbau – Konzept zum Setzen von Endhalterklemmen bei der Klemmenleistenmontage. In *Automation 2018* (pp. 95–108). VDI Verlag. <https://doi.org/10.51202/9783181023303-95>
- Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path aggregation network for instance segmentation. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1803.01534>
- Loop, C. (1987). Smooth subdivision surfaces based on triangles. BibTeX. <https://www.microsoft.com/en-us/research/publication/smooth-subdivision-surfaces-based-on-triangles/>
- Nguyen, H. G., Habiboglu, R., & Franke, J. (2022). Enabling deep learning using synthetic data: a case study for the automotive wiring harness manufacturing. *Procedia CIRP*, 107, 1263–1268. <https://doi.org/10.1016/j.procir.2022.05.142>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2016). PointNet: Deep learning on point sets for 3D classification and segmentation. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1612.00593>
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). PointNet++: Deep hierarchical feature learning on point sets in a metric space. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1706.02413>
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You only look once: Unified, real-time object detection. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1506.02640>
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1506.01497>
- Rodrigues, R. S. V., Morgado, J. F. M., & Gomes, A. J. P. (2018). Part-based mesh segmentation: A survey. *Computer Graphics Forum*, 37(6), 235–274. <https://doi.org/10.1111/cgf.13323>
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1505.04597>
- Scheffler, B., & Bründl, P. (2023). *Electrical and electronic components dataset*. Harvard Dataverse.
- Schneider, L., Niemann, A., Beuing, O., Preim, B., & Saalfeld, S. (2020). MedMeshCNN—enabling MeshCNN for medical surface models. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2009.04893>
- SeleniumHQ. Selenium. <https://github.com/SeleniumHQ/selenium>
- Sharp, N., Attai, K., Crane, K., & Ovsjanikov, M. (2020). DiffusionNet: Discretization agnostic learning on surfaces. ArXiv Preprint. <https://doi.org/10.48550/arXiv.2012.00888>
- Sorkine, O., & Alexa, M. (2007). As-rigid-as-possible surface modeling. In *SGP'07, Proceedings of the 5th eurographics symposium on geometry processing* (pp. 109–116). Eurographics Association.
- Spies, S., Bartelt, M., & Kühlenkötter, B. (2019). Wiring of control cabinets using a distributed control within a robot-based production cell. In *2019 19th International conference on advanced robotics (ICAR)* (pp. 332–337). IEEE. <https://doi.org/10.1109/ICAR46387.2019.8981631>
- Stoidner, M., Bründl, P., Nguyen, H. G., Baechler, A., & Franke, J. (2023). Towards the Digital factory twin in engineer-to-order industries: A focus on control cabinet manufacturing. In E. Alfnes, A. Romsdal, J. O. Strandhagen, G. von Cieminski, & D. Romero (Eds.), *IFIP advances in information and communication technology. Advances in production management systems. Production management systems for responsible manufacturing, service, and logistics futures* (Vol. 691, pp. 80–95). Springer. https://doi.org/10.1007/978-3-031-43670-3_6
- Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3D shape recognition. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1505.00880>

- Tempel, P., Eger, F., Lechler, A., & Verl, A. (2017). Schaltschrankbau 4.0: Eine Studie über die Automatisierungs- und Digitalisierungspotentiale in der Fertigung von Schaltschränken und Schaltanlagen im klassischen Maschinen- und Anlagenbau.
- Tkachenko, M., Malyuk, M., Holmanyuk, A., & Liubimov, M. (2020–2022). Data Labeling Software. Label Studio.
- tpaviot. (2023). pythonocc-core. <https://github.com/tpaviot/pythonocc-core>
- Visual Computing for Medicine. (2014). Elsevier. <https://doi.org/10.1016/C2011-0-05785-X>
- Wang, Z., & Lu, F. (2018). VoxSegNet: Volumetric CNNs for semantic part segmentation of 3D shapes. ArXiv Preprint. <https://doi.org/10.48550/arXiv.1809.00226>
- Zhang, Y., Liang, W., Yuan, M., Xiao, J., Li, J., & Peng, S. (2020). Real-time state recognition of switches on electrical cabinet panel using hybrid visual features. In *2020 IEEE 18th international conference on industrial informatics (INDIN)* (pp. 920–925). IEEE. <https://doi.org/10.1109/INDIN45582.2020.9442167>
- Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3D: A modern library for 3D data processing. ArXiv Preprint. [arXiv:1801.09847](https://arxiv.org/abs/1801.09847).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.