TU WIEN Informatics

Automatisierte Extraktion von Komplexitätsmaßen aus Technischen Zeichnungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Martin Riegelnegg, BSc
Matrikelnummer 00510453

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing.in Lisa Maria Kellner
Dr.techn. Thomas Ortner, MMSc

Wien, 2. Mai 2024

Martin Riegelnegg	Eduard Gröller

TU WIEN Informatics

Automated Extraction of Complexity Measures from Engineering Drawings

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Martin Riegelnegg, BSc
Registration Number 00510453

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing.in Lisa Maria Kellner
Dr.techn. Thomas Ortner, MMSc

Vienna, May 2, 2024

Martin Riegelnegg | Eduard Gröller

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Martin Riegelnegg, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. Mai 2024

Martin Riegelnegg

Acknowledgements

I would like to express my sincere appreciation to my advisor, Eduard Gröller, for his invaluable feedback, guidance, and passion for research, which have been fundamental in shaping this thesis. I would like to extend my gratitude to Lisa Maria Kellner and Thomas Ortner for their support. Furthermore, I would like to thank Lisa Maria Kellner for implementing and training the OCR model for our processing pipeline.

This thesis was developed as part of the AM4Rail research project and I would like to thank our research consortium partners Fraunhofer Austria, Fraunhofer IAPT, Wiener Linen, and ÖBB. It has been an honour and pleasure talking to them and gaining insights through their expertise. Special thanks to Wiener Linien for allowing us to show some of their engineering drawings in this work. Furthermore, a big thanks to my colleagues at VRVis for many interesting discussions.

I would like to express my deepest gratitude to my parents for their constant support throughout my studies. Finally, I extend my sincere thanks to Barbara Palier for her diligent proofreading, continuous encouragement, unwavering support, and never letting me down.

Generative AI tools, including GPT4, Writefull, and Deepl-Write were employed in some parts of this thesis to rephrase sentences, enhancing the clarity and diversity of expressions within the text.

This work received funding from the Austrian Federal Ministry for Climate Action (BMK) under grant number 886461 (AM4Rail). The Austrian Research Promotion Agency (FFG) has been authorised for program management.

Kurzfassung

Eine technische Zeichnung ist eine detaillierte Darstellung eines Objekts, die dazu dient, komplexe Informationen für Konstruktions-, Fertigungs- und Instandhaltungszwecke zu vermitteln. Diese Zeichnungen bestehen in der Regel aus mehreren orthografischen 2D-Ansichten eines 3D-Objekts, die mit Bemaßungsinformationen und Metadaten zu bestimmten Eigenschaften des Objekts versehen sind. In den letzten Jahrzehnten haben sich technische Zeichnungen von handgezeichneten Skizzen zu hoch standardisierten Dokumenten entwickelt, die mit Hilfe von CAD-Software erstellt werden. Die sich dadurch ergebende stilistische Vielfalt erschwert die automatische Extraktion abstrakter Informationen. Additive Fertigung (AM) gibt Unternehmen die Möglichkeit, Ersatzteile für Wartungszwecke nach Bedarf herzustellen. Die Bewertung des AM-Potenzials von Ersatzteilen ist sowohl aus wirtschaftlicher als auch aus technischer Sicht wichtig. Diese Machbarkeitsanalyse erfordert die Interpretation von Komplexitätsmaßen, die aus technischen Zeichnungen abgeleitet werden können. Die Hauptabmessungen eines Objekts sind wichtige Komplexitätsmaße, um eine AM-Potenzialanalyse durchzuführen. In dieser Diplomarbeit entwickeln wir eine Verarbeitungspipeline, welche die Extraktion von Komplexitätsmaßen aus technischen Zeichnungen automatisiert. Unsere Methodik basiert auf einer detaillierten Analyse der Eigenschaften technischer Zeichnungen aus unterschiedlichen Epochen. Außerdem versuchen wir, die einzelnen Schritte unserer Pipeline interpretierbar zu machen. Wir segmentieren wichtige Komponenten der Eingabezeichnung, um mögliche Bemaßungslinien zu identifizieren, die dann durch eine Abfolge von Bearbeitungsschritten verfeinert werden. Das Ansichtsraster wird so bestimmt, dass wir jeder Ansicht Achsenbeschriftungen zuweisen können. Wir verwenden OCR, um die erkannten Bemaßungszahlen auszuwerten. Anhand der OCR-Ergebnisse ermitteln wir das Verhältnis zwischen den OCR-Werten und der Länge der Bemaßungslinien in Pixeln. Zur Demonstration der Leistungsfähigkeit unserer Pipeline implementieren wir einen Forschungsprototypen. Die Ergebnisse einer quantitativen und qualitativen Evaluierung bestätigen die Effektivität unseres Ansatzes zur automatischen Extraktion abstrakter Informationen aus technischen Zeichnungen.

Abstract

An engineering drawing is a detailed representation of an object used to communicate complex information for the purposes of design, manufacturing, and maintenance. These line drawings typically consist of multiple 2D orthographic views of a 3D object, along with dimensioning information and metadata about specific properties. Over the past decades, engineering drawings have evolved from hand-drawn sketches to highly standardized documents created with the help of CAD software. The large variety of engineering drawings makes it difficult to automatically extract abstract information in a robust way. The emergence of additive manufacturing (AM) promises companies that they can produce spare parts on demand for maintenance, potentially increasing the operational time of their infrastructure. Evaluating the AM potential of spare parts is essential from both an economic and technical perspective. This analysis of economic and technical viability requires the interpretation of complexity measures that can be derived from the engineering drawing of a spare part. The external dimensions of an object are key complexity measures to facilitate an AM potential analysis. In this thesis, we propose a processing pipeline that automates the extraction of complexity measures from engineering drawings, focusing on the external dimensions of the depicted objects. An in-depth examination of engineering drawings from different eras forms the basis of our methodology. Our pipeline is designed to be adaptable and consists of interpretable stages for specific tasks. We segment important entities in the input drawing to detect candidate dimension lines that are subsequently filtered by a sequence of processing steps. The grid structure of the orthographic views is determined, which allows us to assign axis labels to each view. We run optical character recognition (OCR) on detected dimension numbers and use the results to optimize the ratio between the OCR values and the length of dimension lines in pixels, providing us with a solution that is resilient to errors in the OCR predictions. A prototypical implementation of our pipeline demonstrates its capabilities in handling a large variety of drawings. We conduct a basic quantitative and qualitative evaluation of our methodology. The results confirm the effectiveness of our approach in automatically extracting abstract information from real-world engineering drawings.

Contents

CHAPTER 1

Introduction

1.1 Motivation

This thesis is written as part of the AM4Rail research project, which is carried out by a consortium of institutions including ÖBB [oeb], Wiener Linien [wl], Fraunhofer Austria [fhaa], Fraunhofer IAPT [fhab], and VRVis [vrv]. The research project is funded by FFG [ffg] and aims to develop a system for performing an additive manufacturing (AM) potential analysis of spare parts in the context of rail transport. This potential analysis covers various aspects, including an assessment of the technical and economical feasibility of manufacturing a given part using AM processes, such as 3D printing. Failures or the lack of certain components in trains or other railway infrastructure can cause delays and cancellations, which can result in significant financial losses for the operating company. Therefore, they strive to minimize the amount of time that their systems are not operational. AM promises railway companies the opportunity to manufacture required parts on demand, in contrast to conventional manufacturing techniques, where they often have to order a large batch of a particular part that involves long waiting times. They may use only a few pieces of the batch and would need to store the rest, increasing the costs for transport and storage. Railway companies want to replace faulty or missing components as quickly as possible, but also try to avoid excessive storage costs for spare parts that may not be needed frequently. Innovations in AM techniques make it possible to produce a wide variety of spare parts while complying with numerous technical and safety requirements. The technical specifications of a spare part are typically defined in an engineering drawing (ED), a type of line drawing that contains comprehensive information about the respective object, such as various views and detailed dimensions. Railway companies need to maintain infrastructure for a long period of time, in some cases for many decades, which is why they possess EDs of spare parts from different epochs ranging from older designs drawn by hand, that are still relevant today, to modern computer-aided design (CAD) examples. The AM potential analysis should be able to

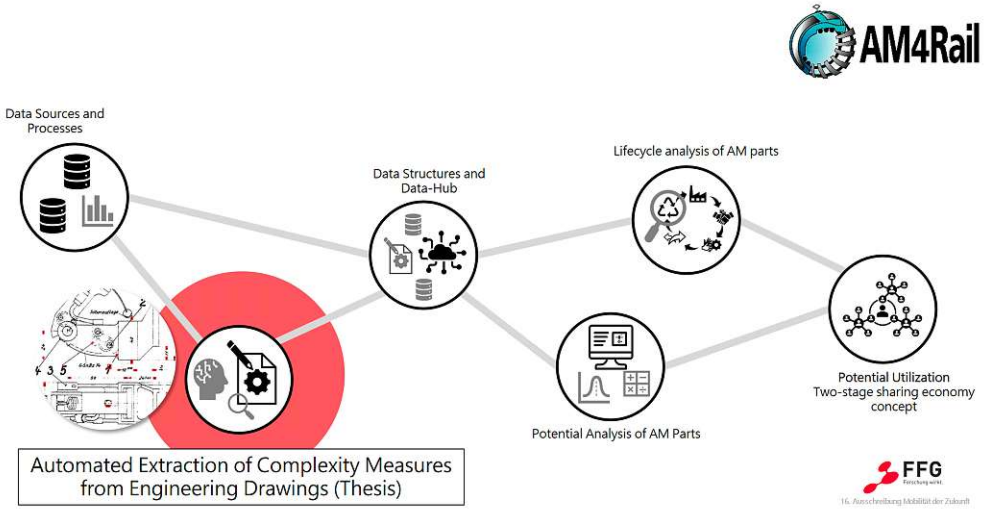


Figure 1.1: AM4Rail project stages. The second stage, which is the work we present in this thesis, is circled in red. Image source Stephan Keckeis, Fraunhofer Austria [fhaa].

handle this large variety of different drawing styles to produce a realistic estimation of the viability, but also of the drawbacks involved when manufacturing a part in an additive way. Ideally, it should process EDs automatically or with minimal human intervention, because railway companies typically have an extensive catalog of spare parts, often containing tens of thousands of EDs in their databases. AM4Rail consists of the following six stages to achieve this goal:

1. Data sources and processes
2. **Automated extraction of complexity measures from engineering drawings**
3. Data structures and unified data hub
4. AM potential analysis of parts
5. Lifecycle analysis of AM parts
6. Two-stage sharing economy concept for potential utilization

Figure 1.1 shows an overview of the AM4Rail stages. The second stage, "Automated Extraction of Complexity Measures from Engineering Drawings", is the work we present in this thesis. An ED contains critical information to analyze the AM potential of the depicted object. This includes the title block, the dimensions, surface finish symbols, and welding symbols. The title block typically contains important specifications, such as material properties, and is of great interest to domain experts for further analysis. The external dimensions of the object define its extent in the three main directions X, Y, and

Z. They are a crucial complexity measure because they determine whether an object fits in the building chamber of an AM machine like a 3D printer and allow domain experts to estimate the cost of the manufacturing process in combination with the material requirements. Surface finish symbols and welding symbols provide domain experts with clues about the necessary postprocessing steps required to obtain the finished object. Therefore, our work in this thesis is a critical step in achieving the objectives of the AM4Rail research project. Furthermore, it is a relevant topic due to the increasing adoption of AM methods in many different fields and the need to reliably interpret EDs from various time periods.

1.2 Problem Statement

The automatic extraction of information from EDs and its interpretation is a challenging problem [ZCK+22], in particular when dealing with stylistically diverse real-world data. In this thesis, we propose a processing pipeline for the extraction of external dimensions from EDs originating from the European railway industry. In addition, we implement a prototype of our pipeline to demonstrate its capabilities in handling a wide variety of drawings and we conduct a basic evaluation. The proposed pipeline should generalize to the characteristics of hand-drawn and CAD generated EDs, including evolving standards, evolving design tools, drawing inaccuracies, and digitization artifacts. The proposed system should be able to process EDs fully automatically and without human intervention because of the large quantities of drawings that a typical railway company needs to assess. Furthermore, the pipeline should produce explainable results despite the use of artificial intelligence (AI) algorithms. It should also be modular, allowing users to exchange specific stages of the pipeline with algorithms better suited to their data distribution without the need to retrain the entire system.

In related work, contextual information has been found to be critical for correctly interpreting EDs [ZJY+23], which is why using a method like optical character recognition (OCR) on its own is insufficient to robustly extract abstract information from EDs, such as the external dimensions. Our method addresses this problem by detecting multiple entity types in EDs and using them in combination to derive the final result. In this thesis, we focus on obtaining the external dimensions of an object depicted in an ED, which is a key complexity measure according to our domain-expert research partners. Figure 1.2 shows an ED with the targeted external dimensions of the depicted object marked with green boxes. The expected results in the three main directions are shown at the bottom of the figure. Additionally, we detect textfields, including title blocks, and surface finish symbols for further analysis by domain experts. Although there are additional important complexity measures, such as volume or curvature, processing them is beyond the scope of this thesis.

This thesis specifically targets drawings that adhere to the international ISO standard. We acknowledge that there are many similarities with other standards, such as the American ASME standard. Because we only have European railway companies as our

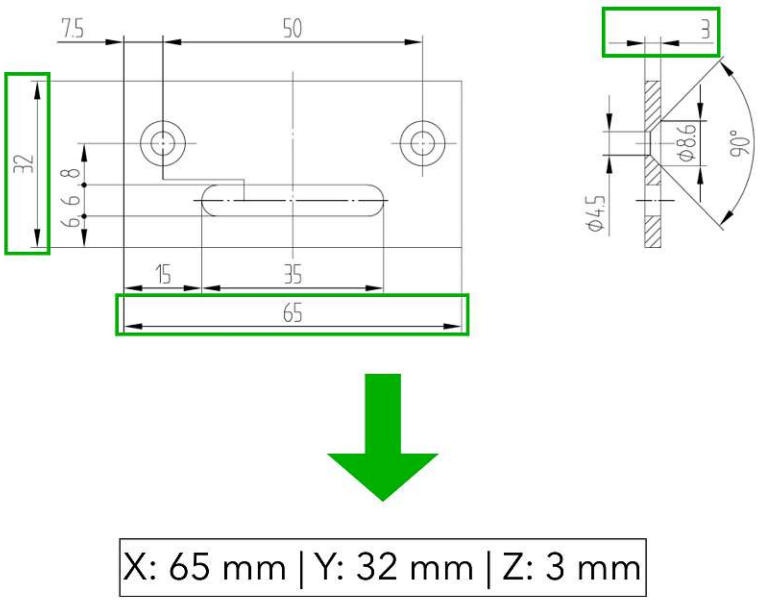


Figure 1.2: The main goal of this thesis is the automatic extraction of external dimensions from EDs. The external dimensions of the object depicted in this figure are marked with green boxes and the expected result is shown at the bottom. ED source Wiener Linien [wl].

partners, we do not have access to ASME data. Due to the unique characteristics and differences between ASME and ISO drawings, we do not take them into account in the design of our pipeline methodology and consider standards other than ISO to be beyond the scope of this work. Furthermore, we only evaluate our pipeline on drawings originating from the European rail industry.

In this thesis, we will attempt to answer the following research questions:

- **RQ1:** Which features and entities in EDs from various time periods are suitable to develop and train a robust algorithm?
- **RQ2:** To which degree can we automize the extraction of external dimensions from real-world EDs?

Based on these research questions, our contributions are as follows:

- An analysis of ED characteristics and how they evolved over time along with the selection of robust features for automatic processing.
- A processing pipeline to automatically extract external dimensions from EDs.
- A basic evaluation of the proposed pipeline.

1.3 Related Work

In this section we give an overview of related work in the domain of extracting information from EDs. In our literature research, we noticed that related topics were first popular in the 1990s and early 2000s, which we attribute to the transition from drawing EDs by hand to designing them with the help of CAD software tools. This change incentivized the exploration of methods to digitize older drawings so that they could be integrated into a modern digital system. Papers from this time period typically employ geometry processing approaches and heuristics to distinguish and interpret various ED entities, such as arrowheads or dimension sets. Although some methods seem to be outdated due to the rise of deep learning (DL) systems, these older papers still provide us with important information on the characteristics of EDs, in particular hand-drawn EDs that are often not considered in recent publications. We aim to develop a generalizable system that works with old and modern drawings, which is why we are interested in the findings of these earlier papers. From the late 2000s to the late 2010s, there were few publications on related topics. However, in recent years, the topic of extracting abstract information from EDs has become popular again due to the advances and availability of AM facilities. We observe a lot of activity in the research community on automatic interpretation, classification, and processing of EDs, often in the context of quality control or online parts quotation. Most current papers use some form of DL as part of the methodology to extract information. The following paragraphs give a more in-depth overview of related work from the first wave of papers as well as recent publications.

Zhang et al. [ZJY+23] present a framework for automating vectorization and semantic segmentation of 2D EDs to streamline the manufacturing and part quotation process. The authors claim that semantic segmentation of EDs is challenging due to severe pixel sparsity and propose a DL-based system that predicts the semantic type of vectorized components. They start with the vectorization of raster images. This process involves several steps, beginning with skeletonization to reduce the strokes of lines to single-pixel width followed by smoothing to obtain a set of traces. The authors identify three point types that exist in the skeleton based on the number of black pixels that are connected to a given point. End points are connected to one black pixel, passing points are connected to two black pixels, and junction points are connected to at least three black pixels. The point type is determined with a simple 3×3 convolution filter kernel that counts the connections. Figure 1.3 illustrates this concept and shows the kernel they use. They define a trace as an ordered list of connected pixels that starts from an end point or a junction point and try to identify all traces in the image until all connected black pixels are part of traces. The traces are refined by detecting corners and splitting them at these points. Short traces with less than four pixels are removed from the set. In the next step, the authors fit cubic Bezier curves to the traces to obtain a set of parameterized vectors, which they need to construct a graph for segmenting the individual components with a graph convolutional network (GCN). In their graph, the nodes are the vectors and the edges represent connections between the vectors. The authors sample n evenly spaced points on each vectorized component and compute nodal features to use in their

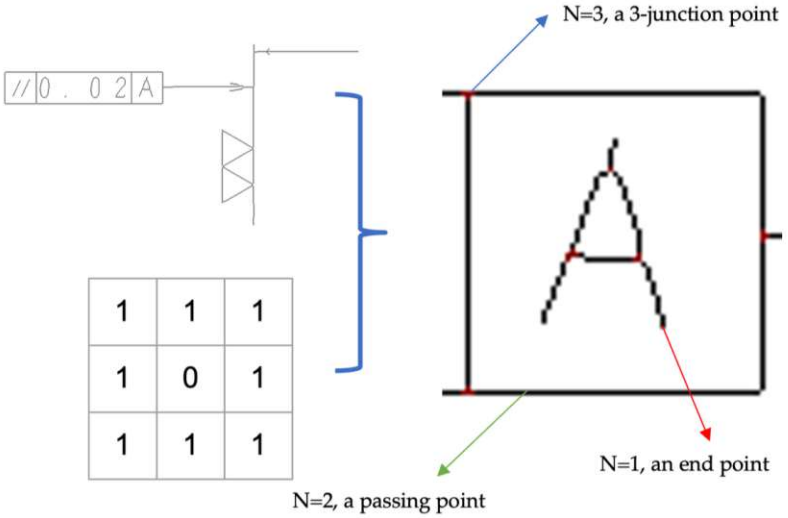


Figure 1.3: Point type classification in skeletonized line drawing as described by Zhang et al. [ZJY+23]. The authors identify three point types on lines based on the number of adjacent black pixels. These types are used to obtain vectorized lines from the original drawing starting and ending at an end point where $N = 1$.

unified graph representation. These features encode characteristics such as shape, length, angle, and curvature. Zhang et al. employ a GCN based on GraphSAGE [HYL17] to segment the individual components. The model is trained on a data set of 430 EDs to predict three classes of components, including object contours, text, and dimension sets. The training set contains only drawings in DXF format [dxf] that are converted to raster images. According to the authors, their best model, called GS5, achieves an accuracy of **90.82**%. They conclude that contextual information encoded in a graph representation improves accuracy. Their approach seems to deliver satisfactory results on modern EDs designed with CAD tools. However, the authors do not address generalizabilty to older drawings of lesser quality.

Intwala et al. [IKCM16] propose an algorithm to detect arrowheads in EDs that is based on the concept of multi-level thresholding. They try to identify and distinguish arrow-shaped features from other graphical entities in the drawing. According to the authors, the method is tested on CAD drawings with varying levels of detail. The input image is converted to grayscale and is subsequently subjected to multilevel thresholding employing the method of OTSU [Ots79]. After thresholding, two morphological operations, black top-hat and white top-hat, are applied to detect arrowheads. The black top-hat operation is useful for identifying solid arrowheads by highlighting dark spots on a lighter background. In contrast, the white top-hat operation is designed to detect line-based arrowheads by accentuating bright lines on a darker background. Next, they apply additional morphological operations, including erosion and dilation, to break potential links between arrowheads and other entities. Finally, they detect contours, calculate the

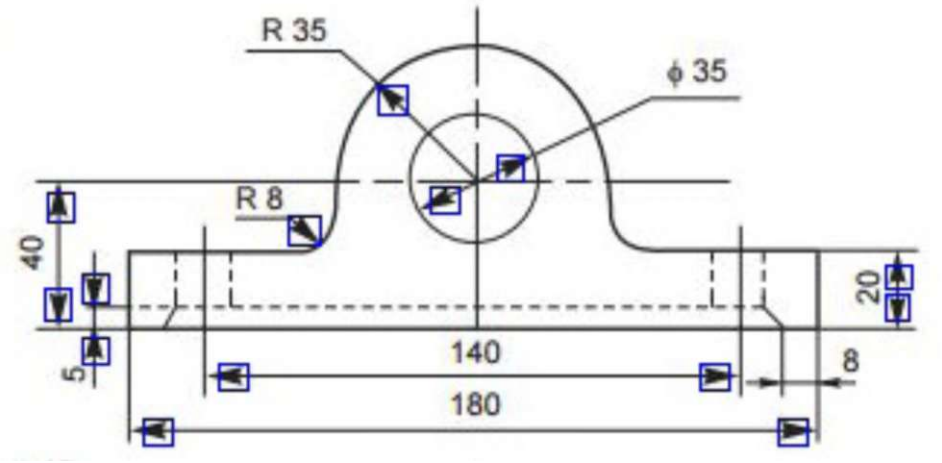


Figure 1.4: Arrowhead detection result from the method proposed by Intwala et al. [IKCM16].

enclosed area, and check if this area falls within a given range of allowed values. If this is the case, the contour and the corresponding region are labeled as arrowheads. The authors choose the range $[3, 70]$ to evaluate the enclosed contour area. They acknowledge that the dependency on image resolution is a limitation of the work. Their method achieves an F1 score of 0.9596, which seems impressive. The authors claim that they evaluated the algorithm on a set of 1199 arrowheads, from which 1156 were detected correctly. However, they do not disclose the number of distinct EDs to which these arrowheads belong. They also do not discuss their data split strategy, so the high F1 score is likely to be biased and probably will not apply to other EDs. Figure 1.4 shows a result of the approach. Although most arrowheads are detected in this instance, the algorithm seems to struggle with the tips of the detected arrowheads, cutting them short in many cases. Furthermore, the algorithm missed two smaller arrowheads to the far right of the image. These arrowheads are still clearly visible and distinguishable from other structures in the drawing, indicating that the algorithm does not generalize well.

Lin et al. [LTH+23] propose a system for detecting components in EDs following the American ASME Y14.5-2018 standard for geometric dimensioning and tolerancing, which is the American counterpart to the international ISO standard. They train the YOLOv7 architecture [WBL23] to detect several entities in EDs and use Google Tesseract [tes] for OCR. The method starts with basic image preprocessing followed by detecting individual views in the drawing. The authors extract image crops based on the view detection results for further processing. Next, they detect five classes of components in the cropped views, including dimensions, tolerances, and feature control frames. OCR is performed on these smaller components to extract textual information and symbols. Lin et al. train three YOLOv7 models for different tasks, including view detection, annotation group detection, and annotation detection. Annotation group detection aims to classify dimensions, tolerances, and feature control frames. Annotation detection targets the recognition of

specific annotations, including geometric dimensions and tolerances. Figure 1.5 shows the layered process for detecting the different drawing components. They use 240 drawings for training and 20 drawings for validation. The annotation group data set consists of 470 views for training and 40 for validation, while the annotation data set contains 285 annotations for training and 20 for validation. The authors do not mention if they perform fine-tuning on the OCR model, but ensure correct text orientation before running the model. They claim to achieve an accuracy of 85 % for view detection, an accuracy of 70 % for annotation group detection, and an accuracy of 80 % for text and symbol recognition. We think that YOLOv7 is capable of detecting these entities, however, it is unclear how the authors would handle ambiguous cases where the views in the drawing are not clearly separated. Furthermore, the method was only trained and tested on clean, modern CAD-generated drawings, which do not necessarily reflect the characteristics of the real-world cases we encountered.

Xie et al. [XLF+22] propose a framework for classifying EDs according to the required manufacturing method in the context of an online quotation process. These manufacturing methods include lathing, sheet metal bending, and milling. They claim that previous methods lack the ability to determine the associated manufacturing process. The algorithm starts with textfield segmentation and removes these structures to simplify further processing steps. The authors train a CascadeTabNet architecture [PGK+20] on a custom data set to obtain tables and textfields in the input image. Next, they perform line thinning and vectorization by tracing black pixels in the remaining contours. This processing step aims to reduce memory requirements and eliminate blank spaces in the drawing. The authors use the Ramer-Douglas-Peucker algorithm [DP73] to remove noisy and short line segments from the vectorized lines. Subsequently, the system removes dimension lines from the drawing because they often contain noisy information that could disrupt the manufacturing method classification according to the authors. They merge collinear lines to simplify the line representations and discard all non-vertical and non-horizontal lines with a tolerance of five degrees. An undirected graph is constructed using the remaining lines, with each line being represented as a node in the graph. The edges are defined on the basis of geometric connections within the local neighborhood of each line employing a k-nearest-neighbor operation. Each node stores the normalized start and end points of the respective line segment. Next, they obtain arrowhead information, including orientation and location, with a pre-trained support vector machine (SVM) [Vap00]. Subsequently, each line includes its proximity to the nearest horizontal and vertical arrowheads. A graph neural network (GNN) analyzes the graph and performs node classification on each node to differentiate the dimension lines. Figure 1.6 shows a dimension line detection result using this method. The detected vertical dimension lines are colored yellow and the horizontal dimension lines green. The dots represent the arrowheads detected by the SVM. The authors use six sequential EdgeConv graph convolutions [WSL+19] in their network architecture. GNNs are prone to information assimilation. The authors propose the use of residual connections across each convolutional layer in their network to alleviate this problem. In addition to detecting dimension lines, Xie et al. use the same GNN to find centerlines

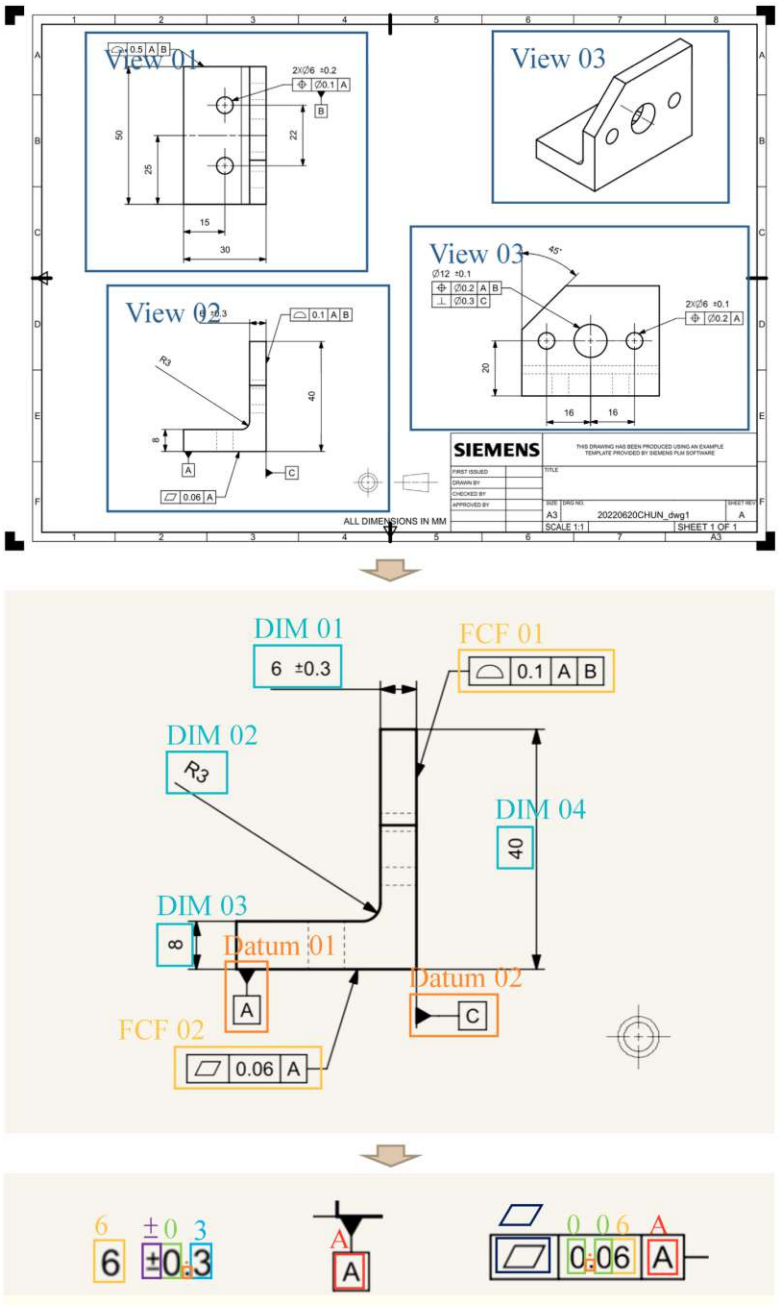


Figure 1.5: Layered entity detection process proposed by Lin et al. [LTH+23]. The authors use three YOLOv7 models to perform different tasks, including view detection (top), dimension and feature control frame (FCF) detection (middle), as well as annotation detection (bottom).

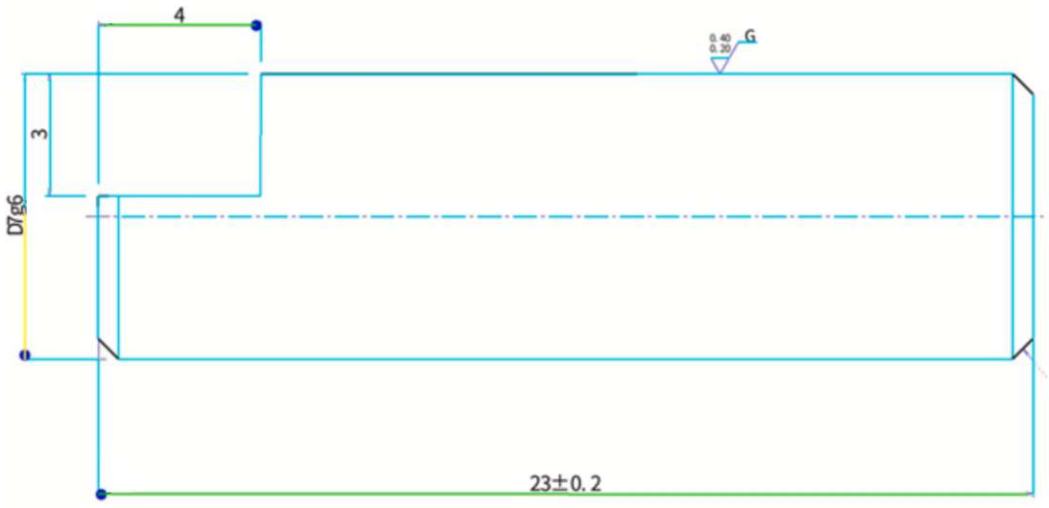


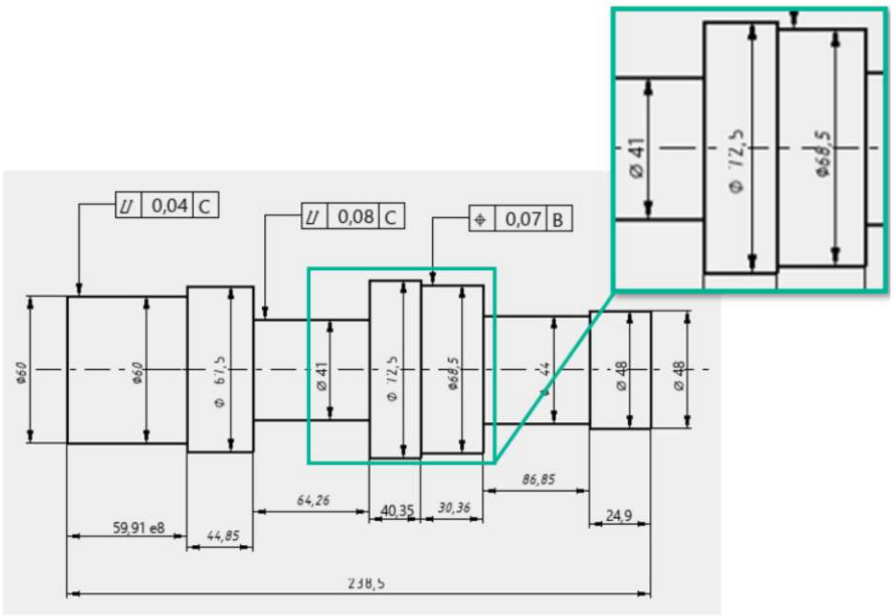
Figure 1.6: Xie et al. [XLF+22] use a GNN to detect dimension lines and to classify the manufacturing process required for a given ED. The detected vertical dimension lines are colored yellow and the horizontal dimension lines are colored green. The dots represent the arrowheads detected by an SVM.

that describe the contours of the object. Afterwards, they use another GNN to obtain the classification of the required manufacturing method for the part depicted in the input drawing. Xie et al. document an accuracy of 96.91% in identifying textfields using their CascadeTabNet. Their line detection approach using an SVM and a GCN achieves an accuracy of 83.2%, while their end-to-end method achieves an accuracy of 90.78% in categorizing the manufacturing process. These results are achieved on a data set of modern CAD drawings. The generalizability of the approach to older drawings, such as those with missing or fragmented lines due to scan artifacts, is not discussed by the authors. Eliminating dimension lines that are off by more than five degrees from the horizontal or vertical axes is not an issue for their purpose, but it would not suffice for our task of extracting the external dimensions from the drawing because scans of EDs are often skewed.

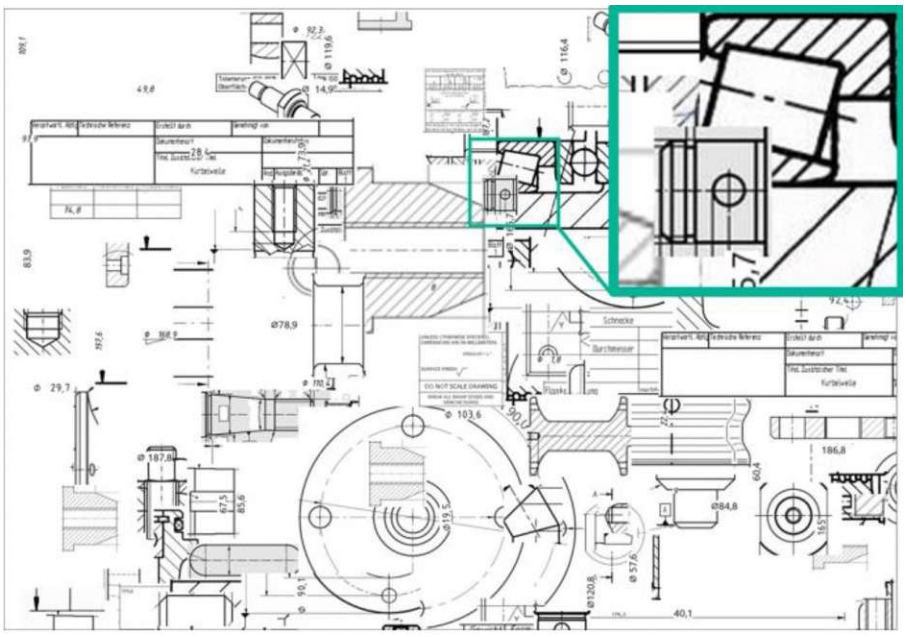
Schlagenhauf et al. [SNH23] propose a system for the automatic detection of text in EDs for the digitization of brown-field processes. Their motivation is the lack of closed CAD/CAM solutions in a sector where EDs are often only available in paper form. The authors claim that there was no specialized OCR model to perform the task of extracting textual information from EDs prior to their work. They mention the recognition of symbols in these drawings as a particular challenge. Off-the-shelf OCR models are not able to recognize these symbols and special characters or other characteristics of EDs. Schlagenhauf et al. note the lack of publicly available ED data sets, as these drawings often contain sensitive company information. To overcome this limitation, they develop a data generator that exploits the structural properties of typical EDs. In addition,

they present an approach for the reliable detection of dimensions, positions, and shape tolerances. The data generator produces synthetic examples based on a set of real drawings to be used for training an OCR detector and recognizer. Schlagenhauf et al. generate two types of artificial drawings. Type 1 drawings show geometries and labels as they would appear in real EDs, whereas Type 2 drawings contain randomly placed elements from nine real EDs. Figure 1.7 shows the two types of synthetic images with (a) Type 1 and (b) Type 2. The authors generate images with a width between 1200 and 1500 pixels and a height that follows standard sheet sizes with a width-to-height ratio of $1 : \sqrt{2}$. They claim to achieve a wide variety of different synthetic drawings that should be suitable to train a detection model. Schlagenhauf et al. also generate another type of image to train the recognition model. For this purpose, they generate smaller images, containing a set of characters that typically occur in real-world EDs. The authors use a Faster-RCNN with a ResNet50 backbone pre-trained on COCO for the detector and only train the last layers of the network. They run several experiments with different data set sizes ranging from 2000 to 7000 synthetic samples. Furthermore, they train another model with only nine real drawings for comparison. This model predicts three classes, including "dimensions horizontal", "dimensions vertical", and "shape and position tolerances". They use the Keras-OCR model for text recognition and train it on 30000 images of the third type. The results show that their data generation scheme is beneficial for training robust models. The method achieves a detection rate of 81.87% and a correct text recognition rate of 79.33%. In comparison, the detection model trained on real samples achieved a detection rate of only 4.67%, while the out-of-the-box Keras-OCR model without fine-tuning achieves a text recognition rate of 46.75%, which is significantly lower than the fine-tuned version trained by the authors.

Zhang et al. [ZCK+22] propose a method for generating EDs to be used as training data. They address the lack of computer support for interpreting EDs automatically. The authors introduce a constrained data synthesis approach that uses only a few manually annotated EDs to produce a virtually unlimited number of training examples. Their method randomizes dimension sets and imposes two major constraints to ensure the validity of the synthetic drawings. This data-driven method is based on a few EDs in DXF format. Zhang et al. start by parsing a DXF file and storing the parametric component information in a unified format. They focus on four entities, including lines, circles, arcs, and dimension sets. In the next step, the authors generate randomized dimension sets in parametric space by using the end points of lines and the centers of circles as key points. In order to preserve the level of data complexity during the randomization of the dimension sets, the total number of linear dimension sets is randomly selected within a range of ± 20 % of the number found in the original drawing. The authors iteratively generate linear dimension sets with a randomly selected pair of key points and a randomly selected orientation that is either vertical or horizontal. They also generate circular dimension sets in a similar way with a circle center and one base point. This algorithm yields a wide variety of examples, however, they might not be realistic enough, which is why the authors impose constraints. The first constraint (C1) ensures that the generated dimension sets do not overlap. The second constraint (C2)



(a)



(b)

Figure 1.7: Two types of artificial EDs generated by Schlagenhauf et al. [SNH23]. (a) Type 1 drawing; (b) Type 2 drawing.

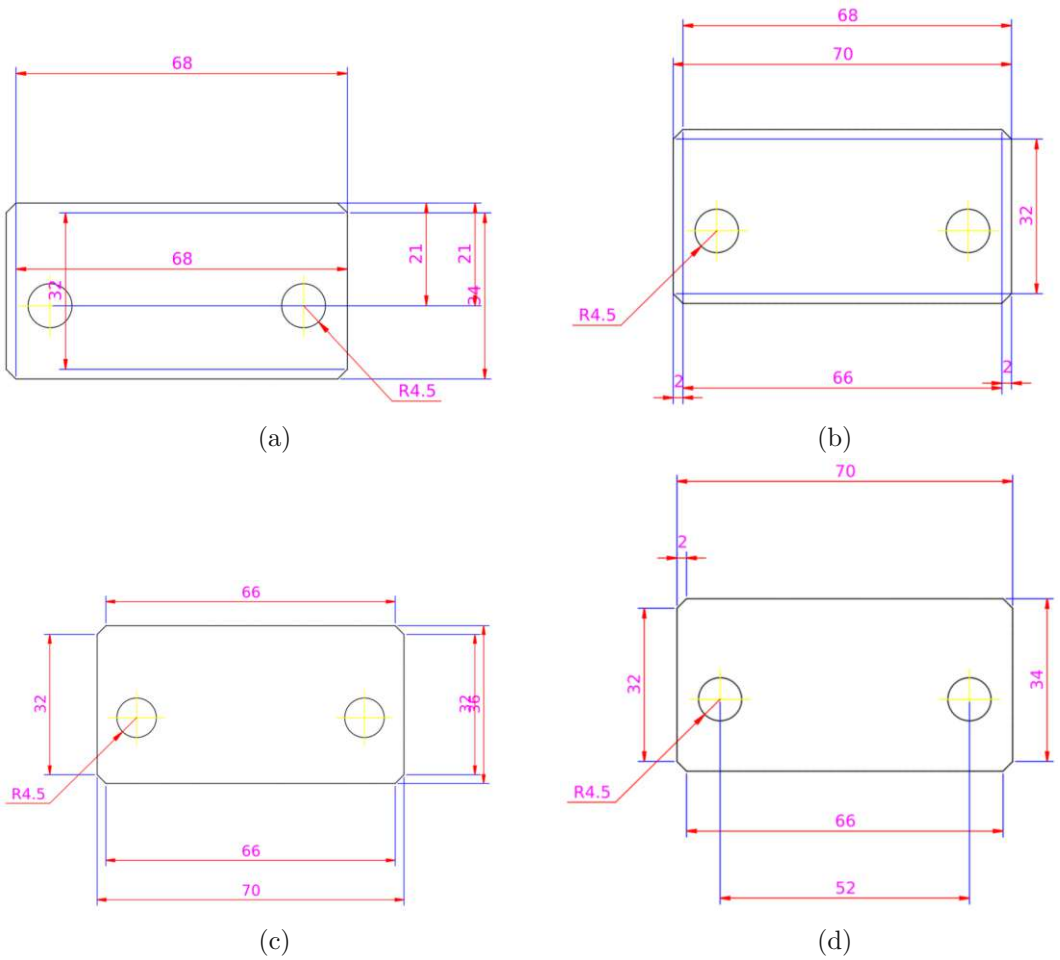


Figure 1.8: Synthetic ED generation proposed by Zhang et al. [ZCK+22]. (a) Unconstrained, (b) C1 only, (c) C2 only, (d) both C1 and C2. C1 ensures that dimension sets do not overlap, C2 enforces that generated dimension sets are located outside of an object view.

dictates that the generated dimension sets should be located outside the contours of the depicted objects, if possible. Placing the longer dimensions outside of all intermediate dimensions is another rule that prevents crossing between dimension lines, but the authors choose not to enforce it during data generation, which, according to them, produces more challenging examples. Figure 1.8 shows four examples of synthetic drawings with randomized dimension sets where different constraints are applied during the generation process. Zhang et al. produce a data set that contains 3200 synthetic drawings based on 32 original EDs. They split the data into a training set with 2500 samples and a validation set with 700 samples. The authors evaluate their data generation strategy by training a multi-layer perceptron (MLP), a decision tree (DT), and a random forest (RF) on the synthetic training set and on the original data. They then evaluate the results obtained

Validation Accuracy %	MLP	DT	RF
Unseen Synthetic	76.84	86.29	87.52
Unseen Original	74.72	82.71	83.78

Table 1.1: Evaluation of the effectiveness of synthetic data generated with the method of Zhang et al. [ZCK+22].

Method	IoU	mAP	Accuracy
Base (U-net)	0.6262	0.6775	0.9937
Base+Ave+CBAM	0.8472	0.8684	0.9942

Table 1.2: Comparing the segmentation performance of a basic U-Net with the architecture proposed by Song et al. [SY22].

from the models trained on synthetic data when presented with unseen original drawings and vice versa, and achieve the results shown in Table 1.1. Furthermore, the authors evaluate the influence of their proposed constraints and the size of the synthetic data set on the performance of the models. The findings indicate that the ability to generalize to unfamiliar geometries increases significantly if the suggested synthesis technique is applied for data augmentation. Moreover, model performance improves logarithmically with the number of synthetic examples, but is limited by an upper bound determined by the diversity in the data.

Song et al. [SY22] describe an approach to segment sheet metal parts in EDs using a U-Net variant. They extend the standard U-Net [RFB15] with convolutional block attention modules (CBAM) [WPLK18] and implement a more complex structure of skip connections in the network, which is the main contribution of the work according to the authors. Song et al. conduct an experiment in which they train the proposed architecture on a non-public data set of EDs of complex welding structures combined with publicly available examples from the United States and Japan. They manually annotate 600 EDs and apply basic offline data augmentation to obtain a total of 4094 drawings for training. The authors compare their architecture, called U-net+Ave+CBAM, with other U-Net variants. Ave stands for average pooling. Examples of segmentation maps produced by the U-net+Ave+CBAM architecture are shown in Figure 1.9. Table 1.2 shows the quantitative results achieved by the different network architectures. It is noticeable that only the values for intersection over union (IoU) and mean average precision (mAP) improve compared to the basic U-Net model.

Ablameyko et al. [ABF+98] develop a system for the automatic recognition of entities in EDs called VEDI (Vectorization of Engineering Drawing Images). They focus on a few main entities, including line types, circular arcs, blocks, hatched areas, and dimensions. VEDI is tested on real EDs and achieves good results according to the authors. Ablameyko et al. start by defining general principles that form the basis of VEDI. The first principle, "from simple to complex" establishes the order in which ED elements are identified, starting with recognizing basic graphic elements and then progressing to more intricate entities.

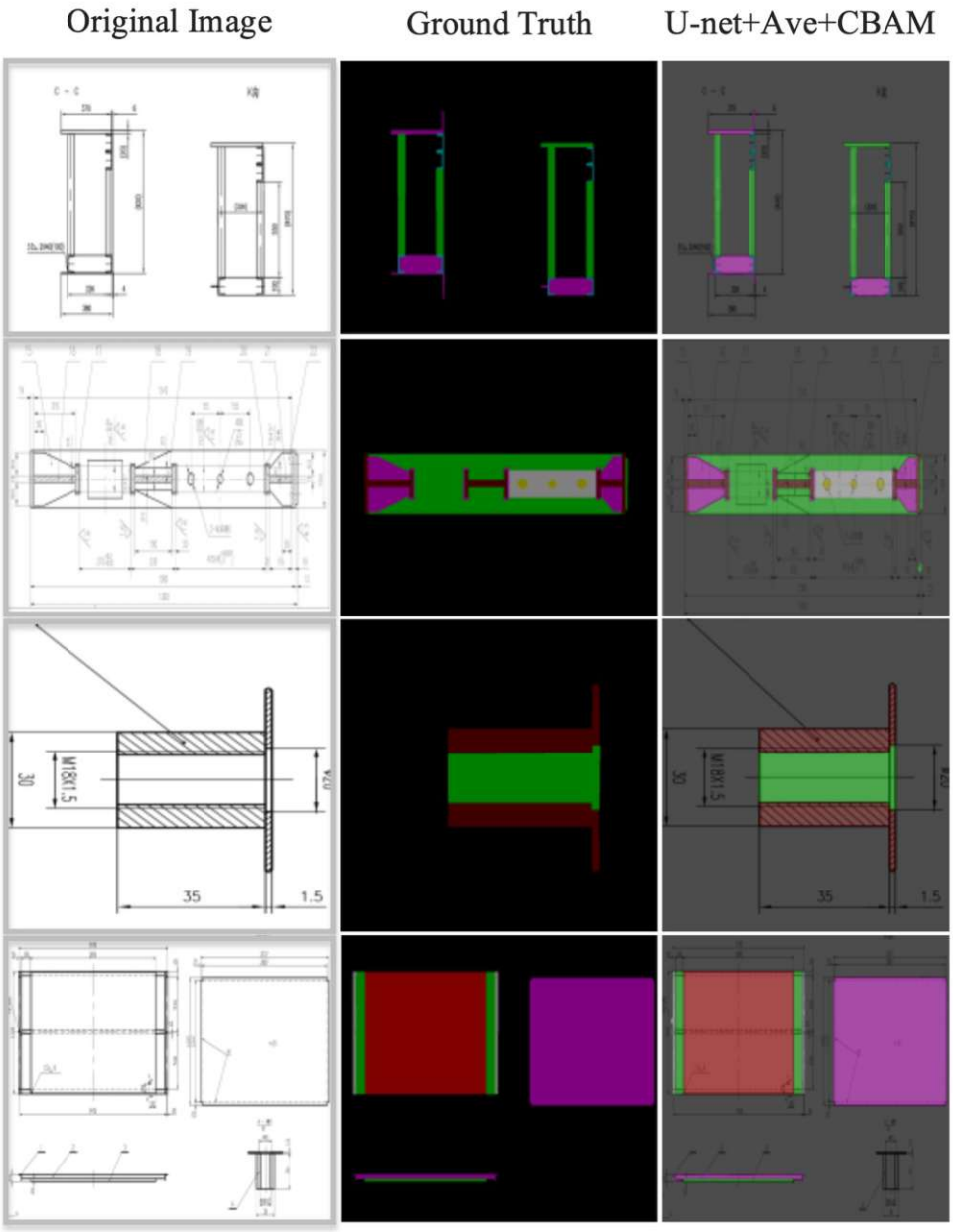
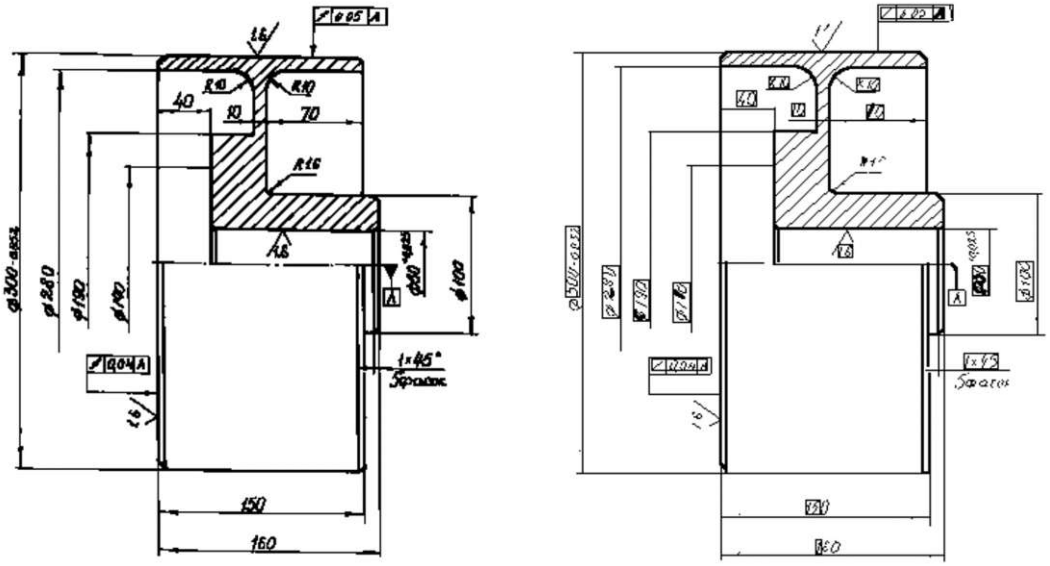


Figure 1.9: Comparison of sheet metal part segmentation results of different U-Net variants by Song et al. [SY22]. The colors represent different segmentation classes.



(a) (b)

Figure 1.10: Ablameyko et al. [ABF+98] develop VEDI, a system for the automatic recognition of ED entities. (a) Input drawing; (b) result after vectorization and recognition.

The second principle establishes rules based on space-logical relations between elements to check for more abstract entities. The third principle, "from local to global analysis" describes a sequential increase in terms of complexity of the recognition algorithms used. The fourth principle emphasizes the exploitation of ED characteristics. The authors aim to collect all relevant information about specific ED entities in the initial stages of processing, before recognition takes place. Recognition involves understanding these unique ED features and their relationships. Next, Ablameyko et al. describe a processing pipeline that takes these principles into account. They vectorize the input ED by line thinning and line segment tracing. Subsequently, they find connected components that form contours and calculate the bounding box, area, and perimeter. This allows the authors to detect text in the image. Afterwards, they remove thin lines from the drawing with morphological operations, which should result in isolated arrowheads that can then be recognized by their distinctive shape. They continue to detect further primitives such as circular and straight line segments before entering the recognition stage of their pipeline. First, they assemble simple graphic primitives based on the vectorized line segments followed by detection of dashed lines, dotted lines, and areas with cross-hatching. They then recognize dimensions by exploiting spatial relationships and characteristics. Subsequently, they combine thick lines into closed contours that describe the boundary of the object. Finally, the authors perform filtering operations, such as aligning near vertical lines to be perfectly vertical. Figure 1.10 shows an input drawing to VEDI in (a) and the result after vectorization and recognition in (b).

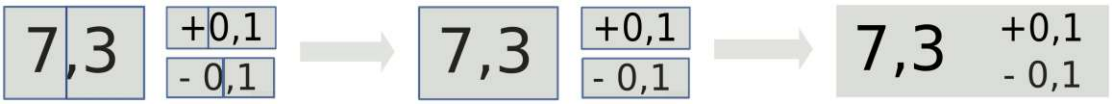


Figure 1.11: Scheibel et al. [SMRM21] use DBSCAN to merge extracted elements into complete dimension sets.

Scheibel et al. [SMRM21] introduce a conceptual approach and a prototype called DigiEDraw to extract dimension requirements from EDs for supporting quality control in production processes. Their approach is based on the DBSCAN clustering algorithm [EKS+96], which the authors apply to merge individual elements of a dimension that belong together. The DigiEDraw method works on PDF files of EDs and consists of three major stages, including preprocessing, clustering, and postprocessing. The preprocessing stage involves converting the input ED into HTML to extract textual and coordinate elements. These elements are then arranged into an array where each sub-array represents a block within the drawing, containing information about textual elements and their bounding box coordinates. In the next stage, the authors utilize DBSCAN to merge extracted textual elements in order to obtain complete dimension sets. They choose DBSCAN because it does not require them to set the number of clusters. The authors address two challenges presented by this approach, which are identifying an optimal distance metric for the results and effectively setting the DBSCAN parameters. The paper proposes a custom distance metric that considers the domain logic and employs an iterative approach to set the required parameters for clustering. This involves defining a distance metric based on the proximity of element bounding boxes, their orientation, and alignment to accurately merge logically connected sets of elements. The postprocessing stage includes cleaning the data using regular expressions to filter out irrelevant information and sorting the elements to ensure that they are in the correct order. This step converts the array of elements into a dictionary format for easier storage and retrieval. An evaluation shows that DigiEDraw can effectively extract dimensioning information from digital PDF EDs. The authors calculate precision and recall metrics for a test set of seven EDs. They achieve precision values ranging from 0.6 to 0.88 and recall values ranging from 0.82 to 0.93. Figure 1.11 shows the clustering of a dimension set that is divided into three blocks and multiple words.

Wendling et al. [WT04] propose a method based on the geometric characterization of arrows, incorporating a comprehensive set of criteria along with the Choquet integral [Cho54] for aggregating these criteria to identify arrows within EDs. Their method assumes that an arrow consists of an isosceles triangle and a connected rectangle. The authors use this concept to formalize several geometric properties crucial for arrow detection. An important property is that an isosceles triangle has a unique angle bisector that bisects it into symmetrical halves and serves as the median for the attached rectangle. This leads the authors to the definition of a global signature that captures the geometric properties of an arrow by integrating local signatures from the vertices of the respective triangle. Wendling et al. define five criteria to be checked, including symmetry, cardinality,

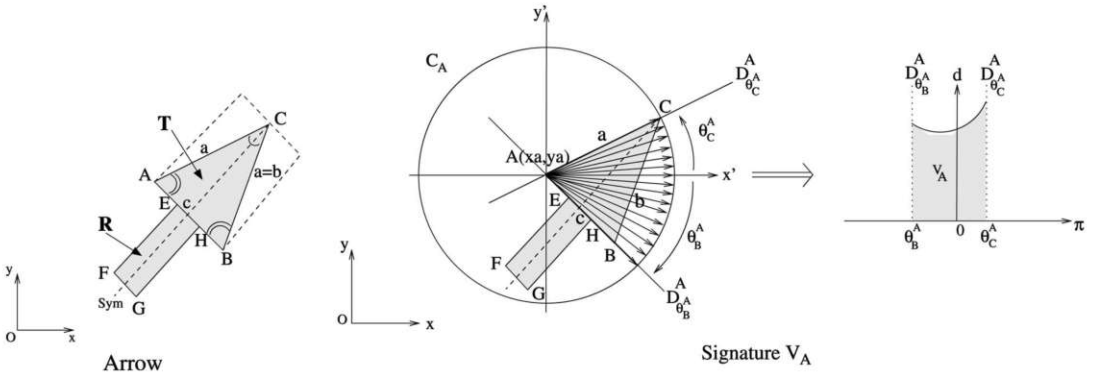


Figure 1.12: Computation of the arrow signature as proposed by Wendling et al. [WT04].

overlap, area, and prototype, each capturing different aspects of the arrow geometry. For example, symmetry evaluates the symmetry of the signature to determine the orientation of the arrow, while prototype evaluates how well the numerical signature of a point triplet aligns with a theoretical prototype, affirming the structural match with an arrow. An essential part of the methodology is identifying triplets of points that potentially form an arrow, addressing challenges such as noise and digitization artifacts. This involves scanning from four main directions to compile point lists, selecting triplets that indicate the geometry of an arrow, and evaluating these against the predefined criteria. The authors employ the Choquet integral to aggregate the defined criteria. This approach is preferable to weighted sums as it can capture the nuances of criteria interactions. Figure 1.12 shows the process of obtaining the discrete signature of an arrow. Wendling et al. test their method on various types of line drawings and achieve satisfactory results, even in the presence of noise or partial occlusions.

CHAPTER 2

Engineering Drawings

In this chapter, we discuss key aspects of EDs and analyze real-world examples from different time periods to identify rules, similarities, and common challenges. Our analysis is based on the examination of real world EDs in collaboration with domain experts. A basic understanding of the structure and features of such drawings is necessary to define requirements for our processing pipeline. EDs are important in the design and manufacturing processes of various industries and serve as a tool for communicating detailed information and specifications of the physical objects they represent. In this thesis, we focus on drawings used in the European railway transportation sector that may have different characteristics than general EDs from other domains, though, most of the core concepts should be similar. We start by introducing the basic building blocks and components of EDs. After that, we examine the style and quality of drawings from as early as 1949 up to today. This analysis aims to discover shared features and norms in visually diverse drawings to serve as the basis for designing a generalizable and fully automatic algorithm to extract the external dimensions of the parts depicted.

2.1 Engineering Drawing Components

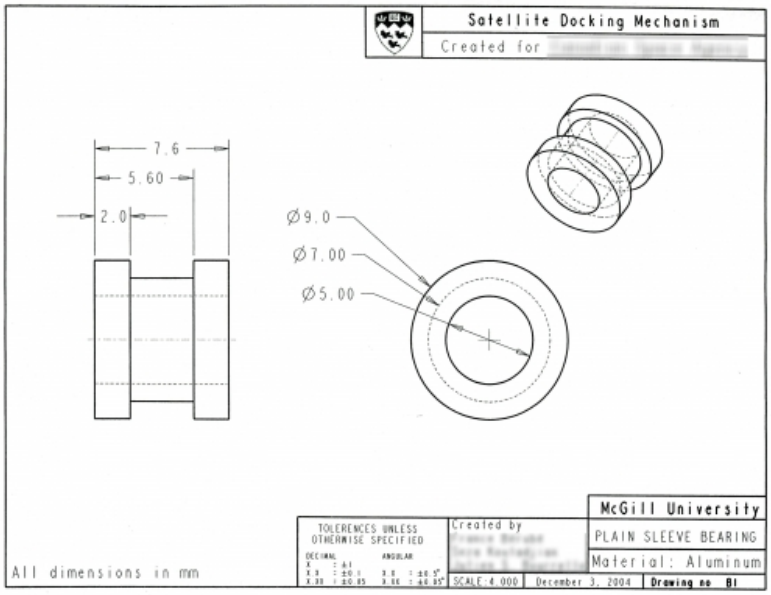
EDs are line drawings of engineering parts and have been critical in the design and manufacturing sectors for a long time. Over the years, EDs have evolved into a highly standardized modality following strict rules and guidelines. While historic drawings are beyond the scope of this thesis, we encounter EDs from various time periods, starting from 1949, as many railway companies still use and maintain legacy parts. The style of EDs has undergone significant change in these 75 years, both in appearance and standards. Drawings created before 1990 are typically drawn by hand, whereas recent drawings are almost entirely drafted with CAD software. We noticed in sample data provided to us by our partners that norms were relatively loose in earlier drawings and became stricter in the 1970s and 1980s. It is also apparent in the sample data that CAD

software has improved significantly over the years. Modern CAD systems produce clean structured images that adhere to various standards, whereas older systems had limited tools, resulting in challenges such as the inability to visually differentiate between different line types. In this work, we only discuss and process EDs converted to a digital image format, for example, by scanning. Current CAD systems have intricate file formats that store additional metadata. However, our partners have limited access to these original files, as manufacturers often only provide them with a digital image of the drawing. There are several types of EDs including detail drawings and assembly drawings. In Figure 2.1 (a) a detail drawing is showing two orthographic views of a single object. Figure 2.1 (b) depicts an assembly drawing of an object that consists of multiple parts and illustrates how to put them together. In this thesis, we deal only with detail drawings because our work should facilitate an AM potential analysis for individual parts.

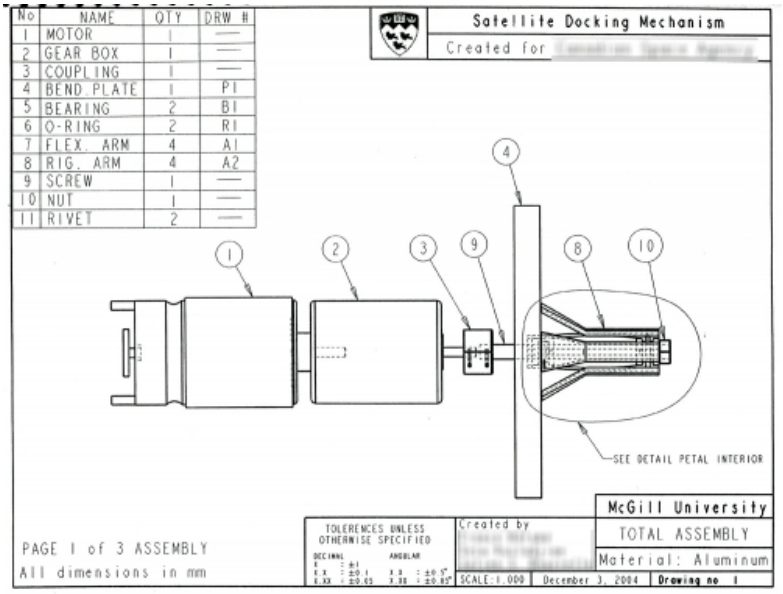
Components of EDs We will now explore the most common components of EDs and their arrangement. The foundation of an ED is a blank drawing sheet, which can have a variety of ISO-standardized sizes, with A4 being the most widely used [mcga]. Objects in EDs should ideally be drawn at a 1:1 scale, so an appropriate sheet size should be chosen. If a scale other than 1:1 is preferable, only integer ratios such as 2:1 or 1:5 should be chosen. The working area on the sheet is typically enclosed by an inside border that contains all other components of the drawing [mcga]. Figure 2.2 shows an example ED where we have labeled some of the most important component classes. The inside border is marked as the blue region that surrounds the entire drawing.

One of the most important components of an ED is the title block, a tabular structure that is typically placed at the bottom right of the working area. Figure 2.2 shows the title block labeled as the red area in the bottom right of the drawing. The position of the title block is not guaranteed, and it can sometimes be mistaken for other tables and textfields in the drawing. The size of title blocks in relation to the sheet size varies significantly, sometimes covering extensive areas, especially in drawings of small objects, and at other times occupying only a small region of the sheet. They typically contain critical metadata about the object depicted in the ED, including the title of the drawing, the drawing number, the type of projection, material properties and finishes, scale indication, estimated weight of the object, the size of the sheet, and the revision history. An example of a title block is shown in Figure 2.3. Title blocks in real-world EDs often have an arbitrary arrangement of table cells, with some cells frequently missing or positioned outside of the actual title block in a secondary table or textfield.

Other main components in EDs are views of the object, which can be seen as the yellow regions in Figure 2.2. EDs can contain a variety of different projections of an object, for example, isometric views as can be seen in the upper right corner of Figure 2.2, or 2D parallel projections, which are the focus in this work. In multi-view orthographic projection, each individual view shows only two dimensions [mcga]. The object is then rotated 90° for the next adjacent view. This gives the viewer a good understanding of the dimensions and 3D structure of the object. The concept of orthographic projection is illustrated in Figure 2.4 (a) where we can observe that the object is projected onto an



(a)



(b)

Figure 2.1: (a) Detail drawing with two orthographic views of a single object. (b) Assembly drawing of an object consisting of multiple parts. Image source [mcga]

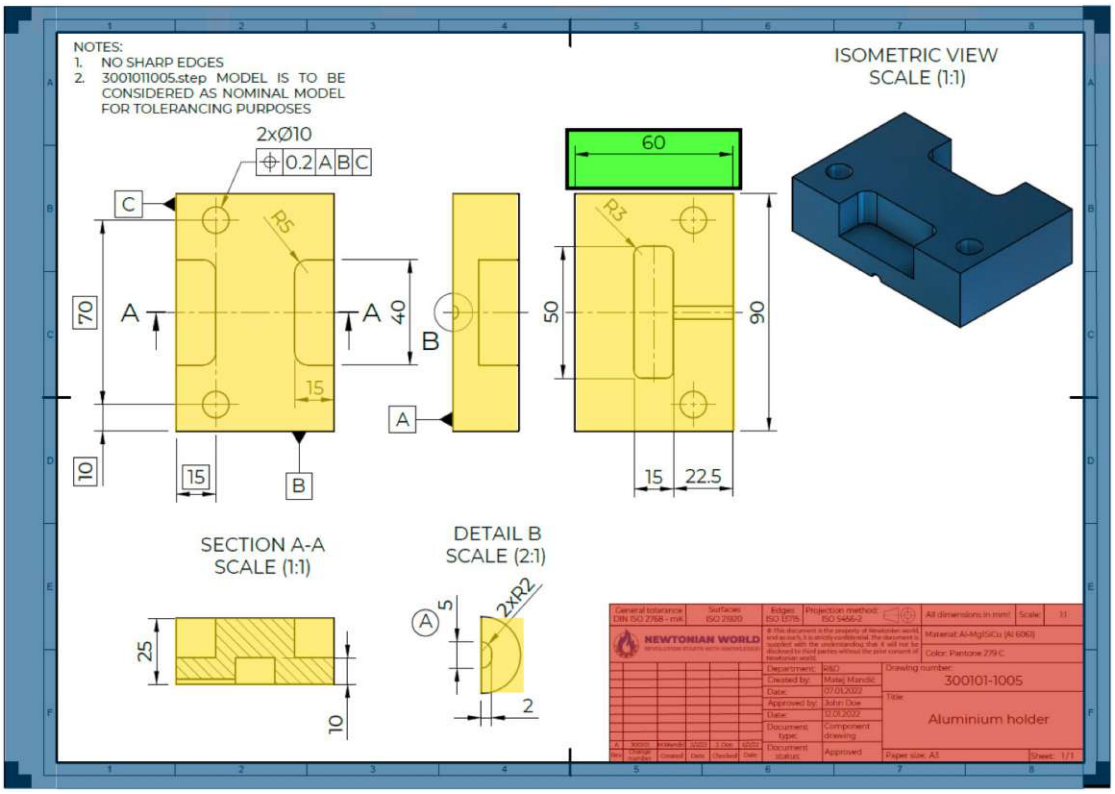
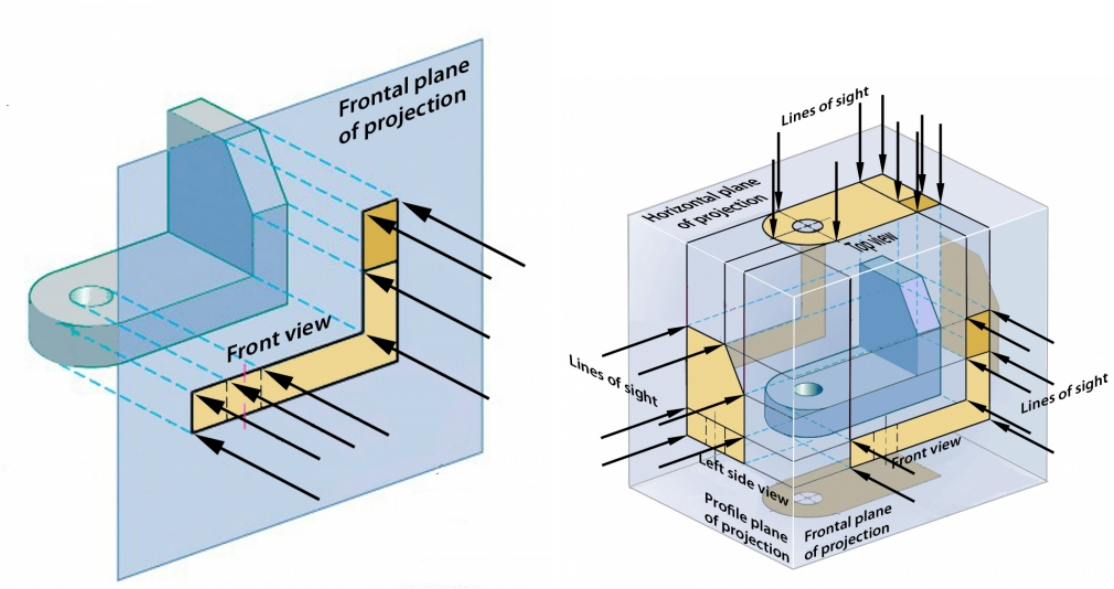


Figure 2.2: Important components of EDs. Inside border surrounding the sheet (blue), title block (red), views of the object (yellow), dimension line (inside the green box), and isometric view (dark blue). Image source [new]

general tolerances DIN ISO 2768-1-m DIN ISO 2768-2-K		scale: 2:1	projection methods DIN ISO 5456-2
	date	name	material: AlMg4,5Mn
drawn	2021.03.03	Dave Jones	
checked			title: Go-kart guard plate
released			
Customer: Tier 1			
WERK24			Drawing-no. 327-7739-2213 / sheet 1 / 1 sh.
			assignment / document no.

Figure 2.3: Title blocks are tabular structures typically located in the bottom right of an ED. They contain important metadata about the object depicted in the drawing. Image source [w24]



(a) (b)

Figure 2.4: (a) 2D orthographic projection of the object onto an image plane through parallel rays. (b) Glass box concept for creating multi-view projections. Image source [mcga]

image plane through parallel rays. Figure 2.4 (b) shows how multiple orthographic 2D views can be obtained by placing the object in a virtual glass box. The sides of this box are unrolled to reveal the grid of 2D views representing the object. It is important to note that not all six possible 2D views need to be drawn. In fact, redundant information should be avoided as can be seen in the arrangement of the view grid in Figure 2.2. The grid structure guides the viewer to understand the spatial relationship between adjacent views. The transition from one view to the next causes a rotation of 90°. Figure 2.5 shows the view grid of the example drawing with views bounded by orange rectangles. Adjacent views are connected with red lines. Moving along a red transition line causes an object rotation of 90° around the axis perpendicular to the transition line. In theory, at least two 2D views are necessary to fully dimension a 3D object. Some drawings only contain a single 2D view of a 3D part where the third dimension is specified either implicitly, for example, diameters in rotational parts, or elsewhere in the drawing, for example, flat parts specified with a material thickness, typically denoted as "$t =$" or "thickness". An example of a fully dimensioned rotational 3D part in a single 2D view can be seen in Figure 2.6. The dimension line on the far right measured as Ø38 specifies the extent of the object in the Y and Z directions.

Special cases include sections and detail views. Sections are often part of the general view grid and show a cut through the object to better illustrate its internals. Detail views are typically drawn to a different scale and show parts of the object enlarged for

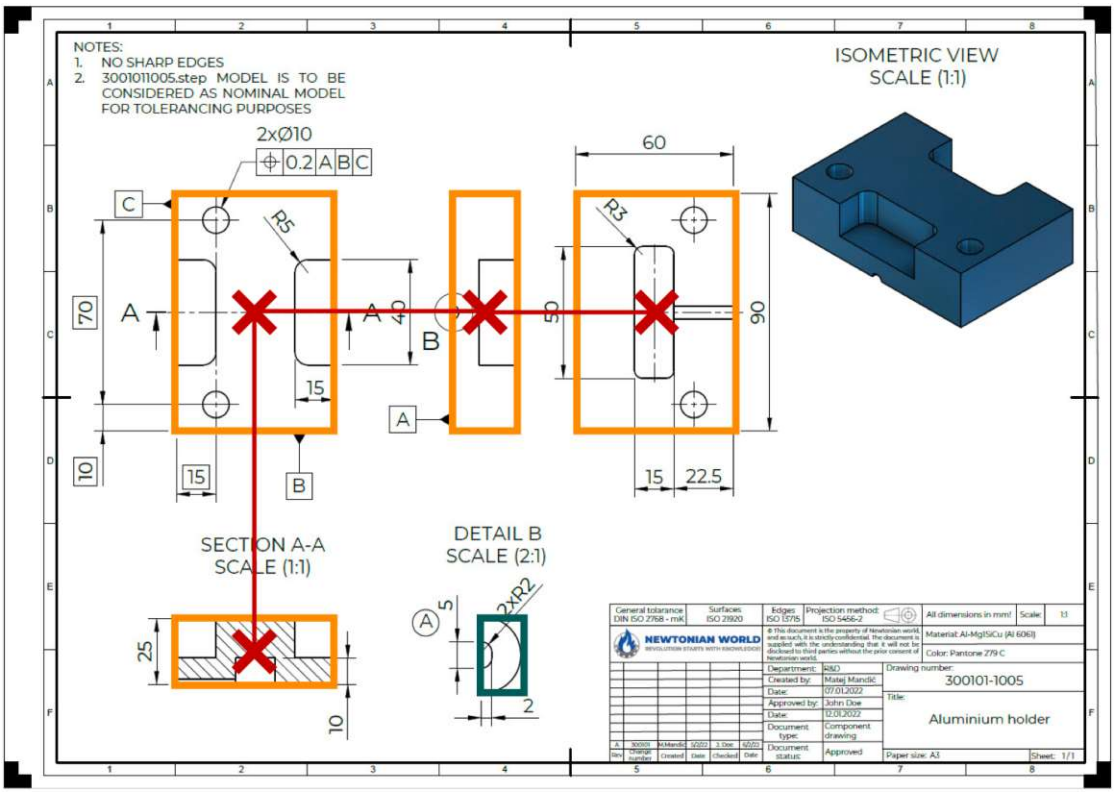


Figure 2.5: View grid of an ED. Object views are bounded by orange rectangles, and adjacent views are connected by red transition lines. Each transition causes a 90° object rotation. Image source [new]

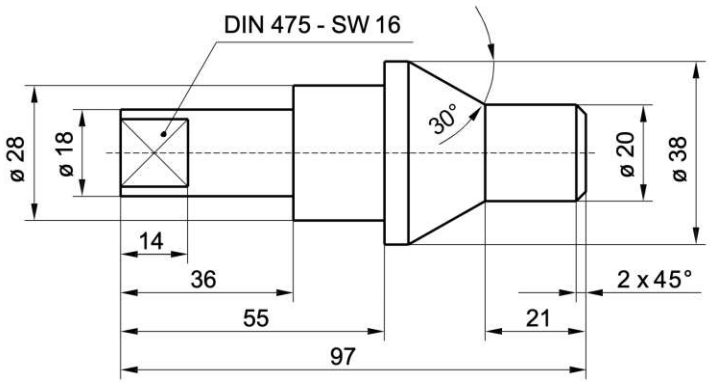
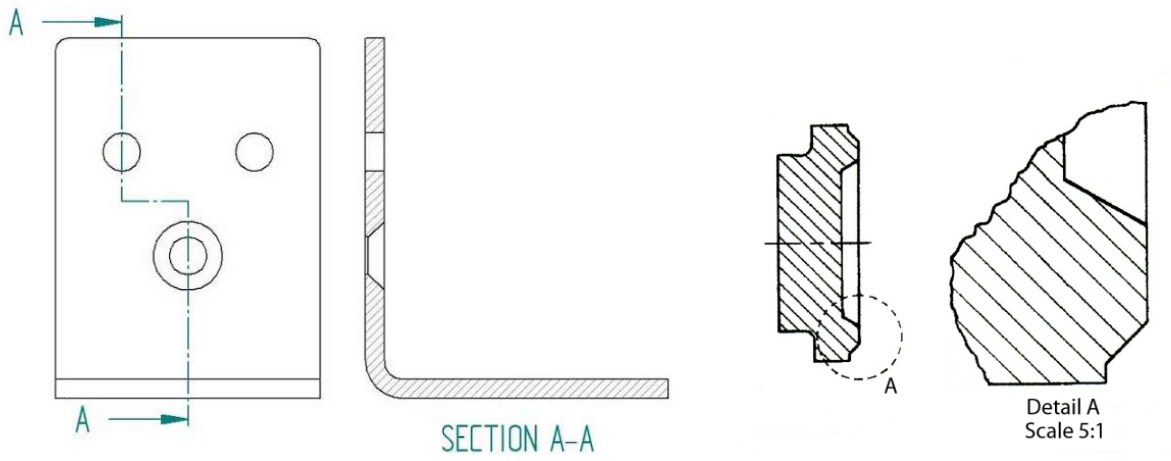
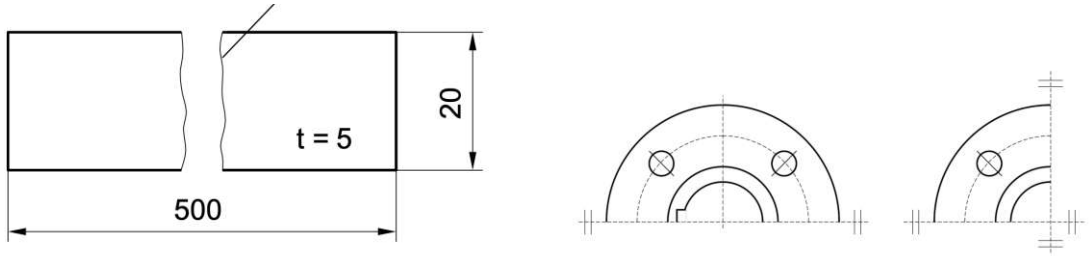


Figure 2.6: Rotational 3D objects can often be fully dimensioned in a single 2D view. The dimension line on the far right (Ø38) specifies the extent of the object in the Y and Z directions. Image source [tum]



(a) (b)

Figure 2.7: (a) Section of an object cutting from A to A. (b) Detail views enlarge certain parts of an object. Image source (a) [fra], (b) [mcga]



(a) (b)

Figure 2.8: (a) Flat part with thickness $t = 5$ and break lines for a more compact representation. (b) Symmetry lines compressing the views of a symmetric object. Image [tum]

better visibility. They should not be part of the view grid because of this scale difference. Figure 2.7 shows an example of a section in (a) and a detail view in (b). The section follows a section line, in this example going from A to A, and the interior of the cut surface is filled with diagonal section lines. The detail view in Figure 2.7 (b) shows an enlarged version of the area covered by circle A. Additional special cases can be seen in Figure 2.8. A flat object with thickness $t = 5$ is shown in Figure 2.8 (a). Due to its large extent in the X direction, the object is split with break lines to make the resulting view more compact. Views of symmetric objects do not need to be drawn completely and can be compressed using symmetry lines, as illustrated in Figure 2.8 (b).

EDs contain a variety of smaller standardized structures that are as important as the larger components. These include lines, arrowheads, and a set of symbols that characterize the object. Figure 2.9 shows the alphabet of lines in EDs [mcga]. Line types are clearly

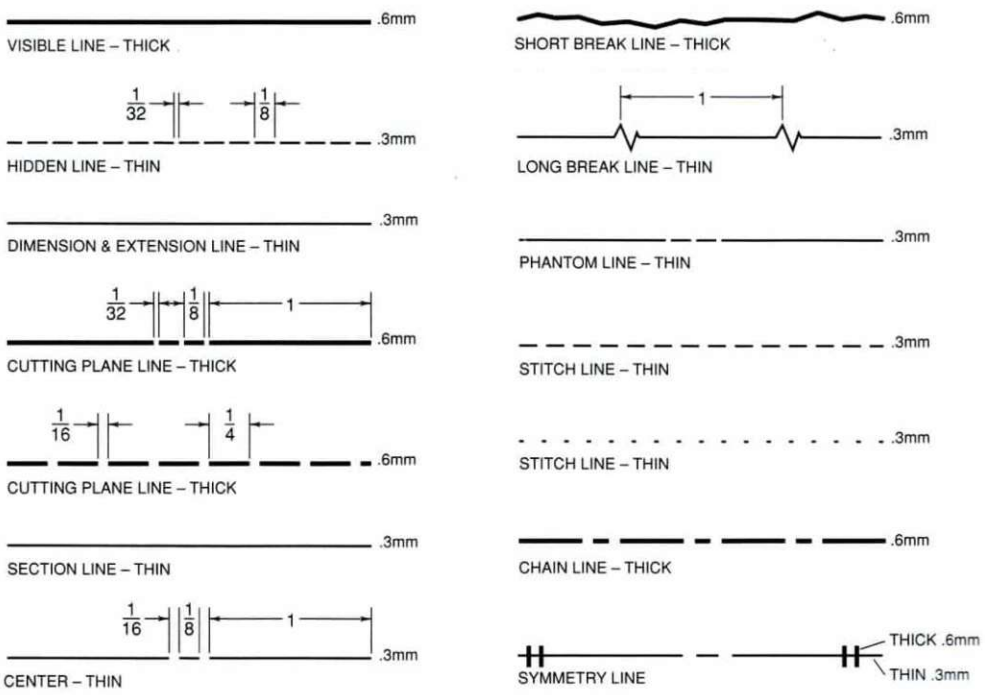


Figure 2.9: Alphabet of lines in EDs. Image source [mcga]

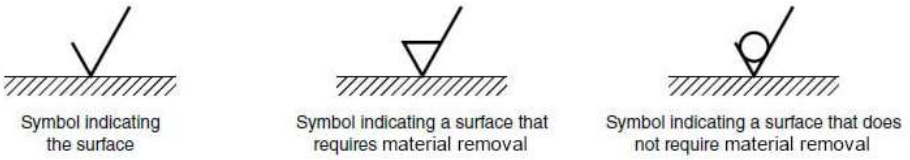
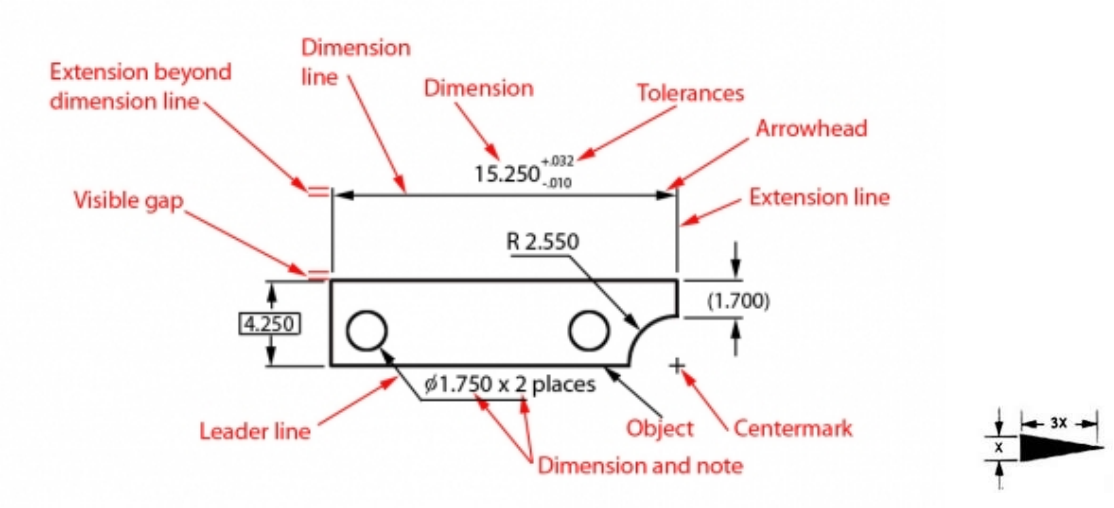


Figure 2.10: Surface finish symbols in EDs. Image source [sur]

specified with different styles and thicknesses to indicate their purpose. For example, visible lines of the object are thicker than dimension and extension lines, while other line types may be dashed. In older real-world drawings, the distinction between line types is often challenging or is only possible by examining the context. Arrowheads are critical for dimensioning objects and should ideally have a length-to-width ratio of 3:1 [ocw]. We call the combination of a line connecting two arrowheads associated with a dimension number a dimension line, an example of which can be seen in Figure 2.2 inside the black box. It is one of the key components that we will try to extract with our processing pipeline in the next chapter. Several types of symbols can be part of an ED, a selection of surface finish symbols can be seen in Figure 2.10.



(a) (b)

Figure 2.11: (a) Various types of dimension information in a view of an ED. Notice the orientations and positions of the dimension lines with respect to the depicted object. (b) Ideal arrowhead with a length-to-width ratio of 3:1. Image source (a) [mcga], (b) [ocw]

Dimensioning of EDs The placement of dimension information in EDs follows a set of clear standards. There are several different types of measurements, including lines, radii, and angles. In this work, we focus on dimension lines. Figure 2.11 shows the types and placement of various dimensioning elements in a view. It is noticeable that the positions and orientations of a dimension line and its associated elements are governed by the object view to which they belong. Ideally, dimension lines run perfectly vertically or horizontally besides the object view and avoid intersections with it. The two arrowheads denoting the ends of a dimension line point in opposite directions. Extension lines run perpendicular to the dimension lines, emanating from the tips of the corresponding arrowheads, and indicate which features of the object are associated with the measurement [mcga]. Dimension numbers are typically placed near the center point of the respective dimension line. In Europe, the values are given in millimeters or meters, while in American drawings, the dimensions are typically measured in inches [mcga]. A common arrangement of dimension lines is staggering the dimension numbers on several parallel lines [mcga] as is shown in Figure 2.12. Moreover, the figure also shows that it is preferable to attach dimension lines to a common point for better readability. A final arrangement of dimension lines we want to address are chained dimensions, when a series of dimension lines is applied on a point-to-point basis [mcga] as can be seen in Figure 2.13.

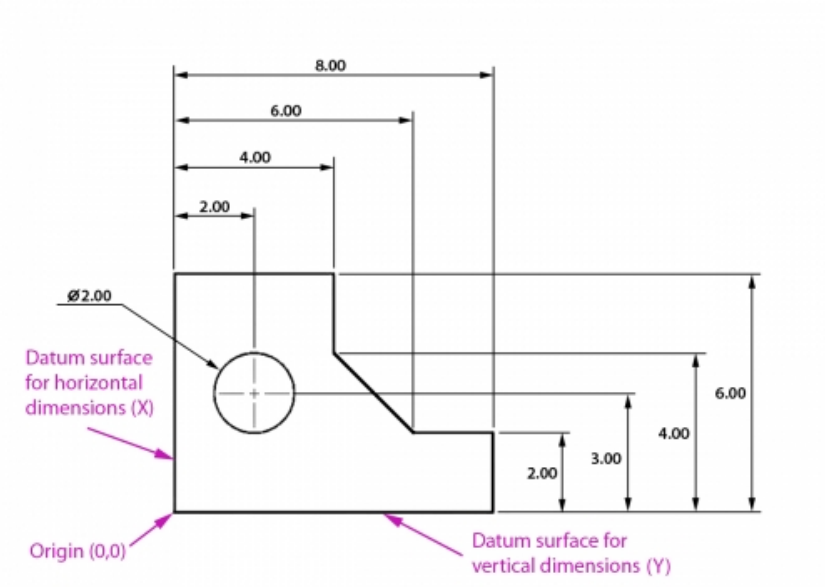


Figure 2.12: Staggered dimension numbers on parallel dimension lines attached to a common point (origin). Image source [mcga]

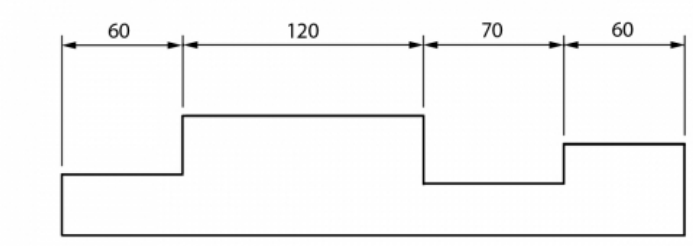


Figure 2.13: Chain dimensioning, a series of dimensions is applied on a point-to-point basis. Image source [mcga]

2.2 Real-World Drawings and Feature Selection

In the previous section, we examined several standards used in ideal EDs. Although modern CAD drawings comply with all of them, older drawings, particularly hand-drawn ones, often differ significantly in the way these standards are used. Some norms evolved over time and were maybe interpreted more loosely, while other rules were changed or newly introduced. It is noticeable that the overall quality of older drawings varies greatly, as can be seen in Figure 2.14. There are multiple reasons for these problems, some of them stem from inaccuracies in the drawing process and others from artifacts introduced during digitization. It is important to note that some old drawings display excellent image quality. Drawing inaccuracies include illegible text, misalignment of components, indistinguishable line types, crossed out text or lines, crooked arrowheads pointing in

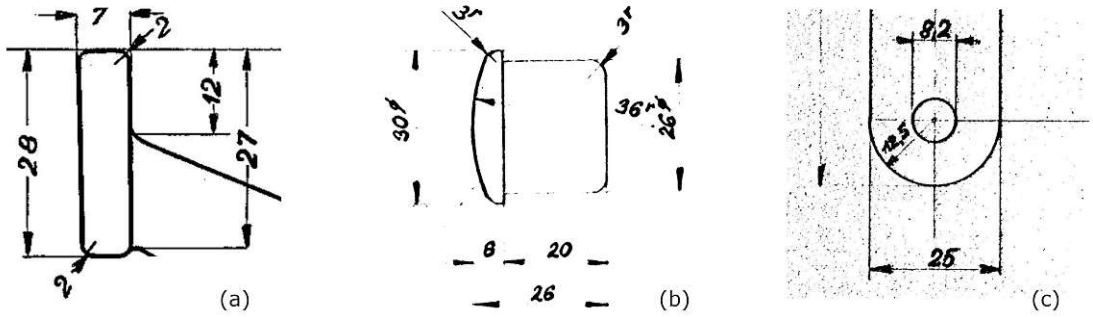


Figure 2.14: Crops from hand-drawn EDs of different visual quality. (a) Drawing from 1949; (b) 1967; (c) 1972. ED source Wiener Linien [wl].

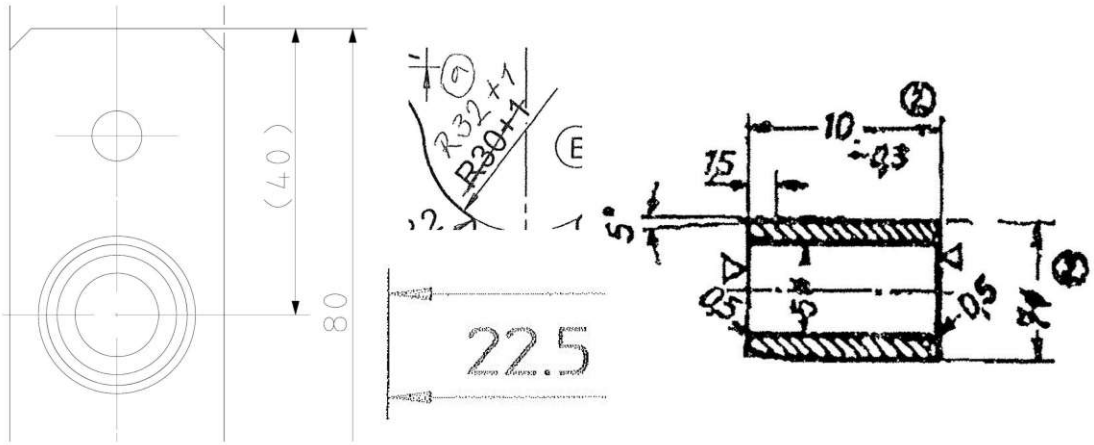


Figure 2.15: Examples of challenging drawings due to inaccuracies such as indistinguishable line types, crossed out or illegible text, scan binarization artifacts, and crooked arrowheads. ED source Wiener Linien [wl].

significantly different directions than the associated dimension line, non-standardized title blocks, non-standard placement of dimensioning elements such as numbers, and many more. Figure 2.15 shows some examples of these challenges in hand-drawn EDs, as well as in drawings produced with early CAD systems. Common digitization artifacts include noise and dirt on the scanned drawing, as well as vanishing lines due to inadequate resolution or binarization errors. The digital EDs we work with come in a wide variety of resolutions, with the largest scans exceeding 200 megapixels and the smallest scans being below one megapixel. We need to take into account all the difficulties mentioned above to improve the generalizability of our processing pipeline, which we design in the next chapter.

Based on our analysis of EDs, we identify some features that are consistent throughout the epochs and drawing styles. They will play a critical role in achieving our goal of

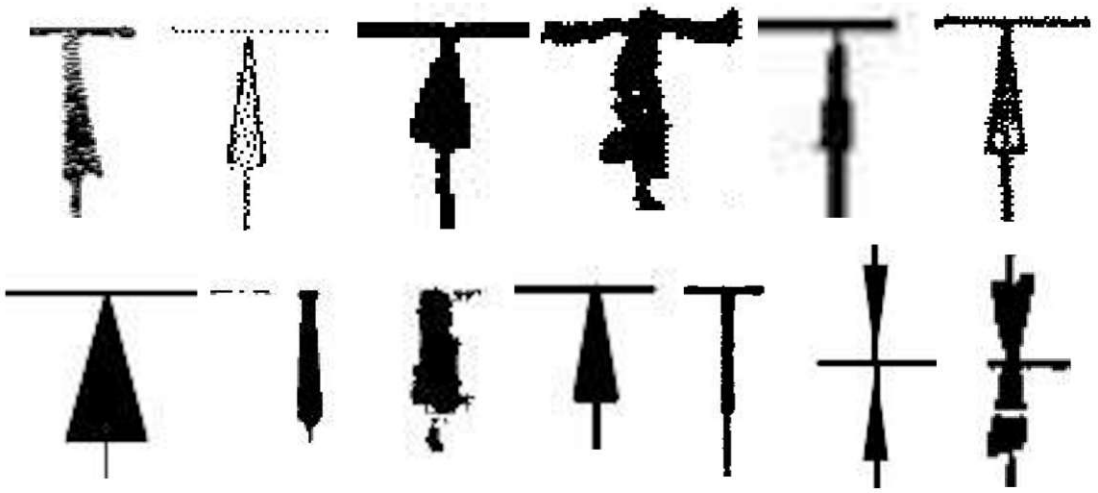


Figure 2.16: A selection of different arrowhead styles we encountered in real-world EDs. ED source Wiener Linien [wl].

automatically extracting external dimensions from EDs. In contrast, some other features seemed to be unreliable. Their appearance may not have been distinctive enough, or they were missing altogether from a significant percentage of drawings in our data set.

Arrowheads Arrowheads are one of the most important structures in EDs and typically have a distinctive shape. Furthermore, their appearance remains relatively stable throughout the drawing styles and time periods, which is why we consider arrowheads to be one of the most reliable features in EDs. They are critical in determining the starting and end points of dimension lines and, therefore, are of great value to our processing pipeline. In some cases, they may have no clear triangular shape, however, they should still be recognizable enough to train an algorithm to detect them in a robust way. Figure 2.16 shows a selection of different arrowhead styles.

Dimension Numbers and Surface Finish Symbols Dimension numbers contain detailed dimensioning information about the depicted object and are necessary features to determine its real size. They are not as recognizable as arrowheads, their shapes may vary and their placement relative to the corresponding dimension lines may change, but they are still distinct enough to be detected reliably. Surface finish and welding symbols are of interest to domain experts for determining the required post-processing effort when additively manufacturing the depicted object. Therefore, we will include them in our pipeline, even though they do not contribute in finding the external dimensions of the object.

Textfields and Views Textfields, such as the title block, contain some of the most valuable information for domain experts. They sometimes cover significant regions of the drawing and are present in almost all drawings we have encountered. Their characteristic appearance should allow us to detect them reliably. Considering textfields with arbitrary

placement and orientation is crucial, as we should not solely rely on their usual position in the bottom right corner of the frame. This is especially important because the drawing might have been scanned at an angle or even in an inverted position. Views are present in all drawings we want to process. They are necessary to determine the rotation grid structure in order to assign the correct axes to detected external dimensions. We will consider a view to contain the 2D object view and in addition all dimensioning information associated with it. Furthermore, we want to accurately segment the 2D object views without the surrounding clutter.

Lines The different line types are important structures in EDs, as can be seen in Figure 2.9. They would be a valuable feature if we were trying to process modern CAD drawings only. However, we noticed in many older EDs that a distinction between them was impossible and that they often vanished altogether due to poor scan quality or image processing artifacts. Some examples of missing and indistinguishable lines can be seen in Figure 2.15, illustrating why we consider them to be unreliable.

Scale Indications We noticed in our analysis of EDs that scale indicators are important but can be hard to detect because they are often placed in arbitrary locations and are missing in a number of examples. In some cases, more than one scale is used in a drawing, making detection and correct interpretation even more unreliable. Therefore, we will not include them in our processing pipeline, but rather infer the scale of the drawing from the detected external dimensions.

CHAPTER 3

Pipeline for Processing EDs

In this chapter, we propose a pipeline for processing EDs. The main goal of this pipeline is to reliably extract the external dimensions of the part shown in the input drawing to facilitate an AM potential analysis. It is important that the system runs fully automatically because railway companies want to analyze a large number of EDs in their databases for their AM potential. Our objective is to develop a pipeline that exhibits strong generalizability across EDs from various time periods and with different characteristics. Instead of relying on a single end-to-end algorithm, the approach aims to make the interaction between different stages of the pipeline understandable, resulting in interpretable outputs. This enables us to select hyperparameters more efficiently, such as choosing specific tolerances, without the need to retrain the entire system. Furthermore, individual stages, such as the segmentation algorithms or the OCR model, can be replaced at a later point with different versions that are better suited to a new data distribution, as long as the interface remains the same. In such a scenario, only a few components of the pipeline need to be fine-tuned to match the new data distribution, making this approach more cost-effective in comparison to a black-box monolithic solution. The analysis of EDs in the previous chapter motivates the design of the pipeline and the methods chosen for each stage. Furthermore, it forms the basis for our data preparation, augmentation, and generation strategies, enabling us to take full advantage of machine learning algorithms for particular tasks despite limited training data. In the following sections, we present a comprehensive overview of our ED pipe and discuss the specifics of each stage, including training, parameter selection, and interaction between stages. The chapter concludes with the robust estimation of external dimensions for parts depicted in EDs by calculating an optimal ratio between the lengths of dimension lines in pixels and the results obtained from OCR on potential dimension numbers. In addition, we provide an overview of further complexity measures that could be derived from the outputs of the ED pipe.

3.1 Pipeline Overview

Figure 3.1 shows a diagram of the proposed processing pipeline. Our ED pipe takes an image of an ED with arbitrary resolution as input (a). We apply basic image preprocessing steps, such as converting the input image to grayscale and normalizing the pixel values to a range between zero and one. In the following step (b), we segment important elements of the image, such as textfields, views, arrowheads, dimension numbers, and surface finish symbols. We utilize these segmented entities for an initial estimation of candidate dimension lines in the image (c). Afterwards (d), we generate a simplified version of the input drawing by removing already known elements from the original image, which is then used to perform fine object view segmentation at multiple scales (e), resulting in relatively accurate object boundaries. Next (f), we connect the identified object views and the previously found dimension line candidates and run a sequence of outlier filtering procedures that remove implausible dimension lines from the candidate set. Subsequently (g), the drawing is analyzed to identify the grid structure of the object views to determine the relative rotation of each object view within the grid. In a typical ED, the object is rotated 90 degrees between adjacent grid cells. This fact enables us to assign two of the three main axes X, Y, Z to each cell on the grid. We then derive the longest dimension line in each of the three principal directions and link these lines to potential dimension numbers (h). OCR is performed on the linked dimension numbers. Since many OCR models are sensitive to the orientation of the input text, we run OCR four times for all possible 90-degree rotations of the potential dimension numbers. We only process small snippets cropped from the original input drawing based on the number segmentation performed in a previous stage to further reduce confusing inputs to the OCR model. Finally, we calculate an optimal ratio between the lengths of the detected dimension lines in pixels and the numerical results from running OCR on the potential numbers. This ratio maps lengths measured in pixels to physical dimensions in millimeters. The ratio depends on the scale of the drawing and on the scan resolution, so most input EDs will have a unique ratio. However, the ratio must be constant in a valid object view grid of an individual ED, which allows us to robustly filter outliers from the OCR results. The external dimensions in X, Y, Z are obtained by multiplying the line lengths with this ratio (i). Furthermore, we can check if the object view boundaries are longer than the selected dimension lines and improve the output by taking the maximum extent in X, Y, Z.

3.2 Data

The data set was provided to us by our partners ÖBB and Wiener Linien. It contains 200 EDs in total that were randomly selected from their databases. The samples were examined by domain experts from our research consortium, who verified that the selection is representative of the distribution of characteristics in real-world EDs that they typically encounter in their work. The samples cover a wide range of different styles, including hand-drawn and CAD drawings. The data set contains EDs from as early as 1949 to

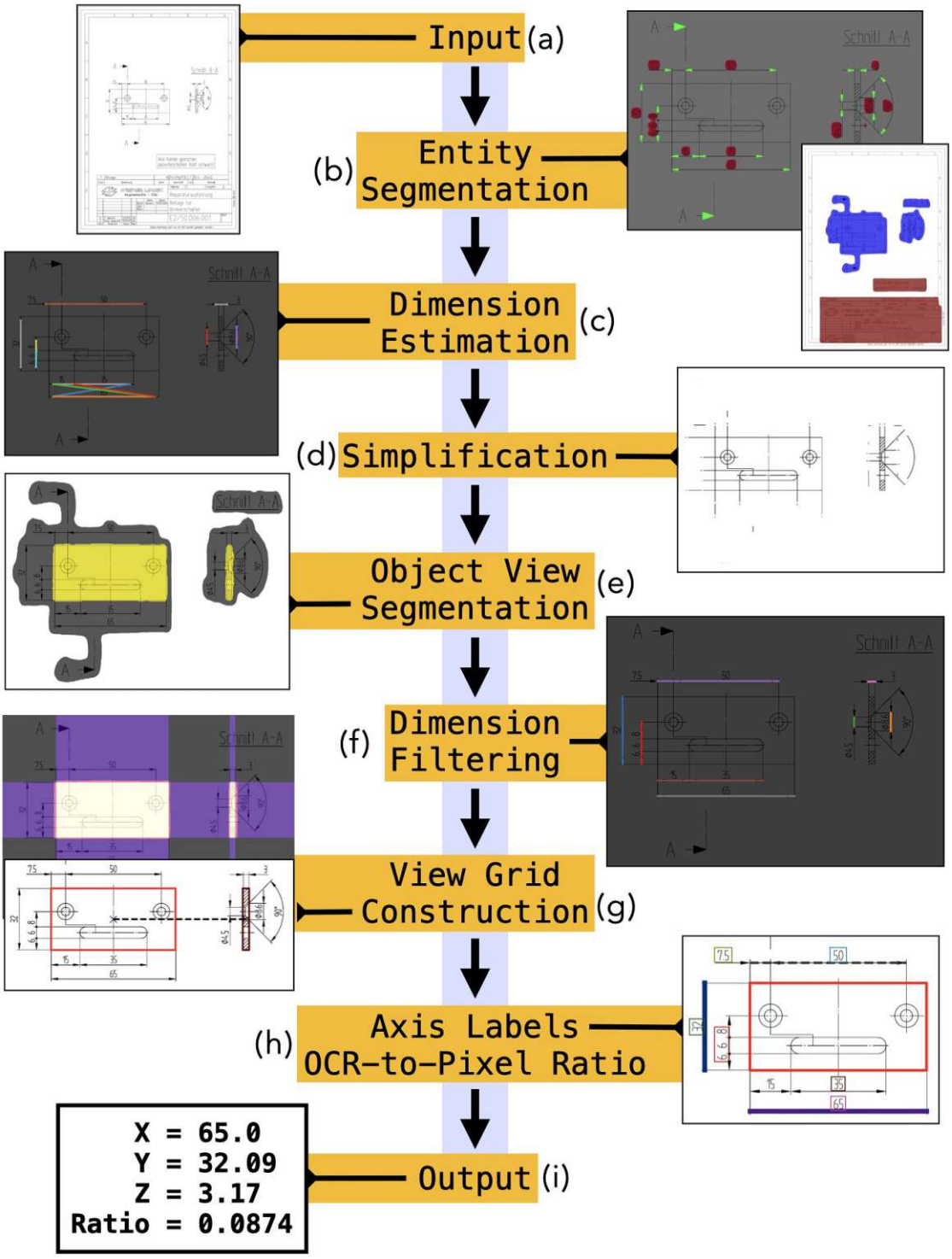


Figure 3.1: Overview of our ED pipe.

more recent drawings from 2023. Furthermore, it includes samples of varying drawing quality, which encompasses many artifacts and defects that we discuss in the data analysis chapter. These artifacts include bad scan quality, vanishing or faint lines, skewed lines, indistinguishable line types, chained or incomplete measurements, illegible text, noise, and many more.

Data Split We split the data set into a training set consisting of 110 samples and a test set consisting of 90 samples. To train pipeline stages that perform image segmentation using neural networks (NN)s, a random subset of the training set is used for validation purposes. The parameter selection in the other stages is performed on the entire training set.

Data Annotation In order to develop, train, and evaluate the pipeline, it is necessary to manually annotate training and test examples. To evaluate the end-to-end performance, we annotate all samples in the test set. The annotations for this task are simple and consist only of the three external dimensions in millimeters for the main axes (X, Y, Z). Additionally, the ED pipe has several segmentation stages that perform particular tasks using NNs, see (b) and (e) in Figure 3.1, as well as an OCR algorithm. Although we train all of these models in a supervised manner, they require different types of annotations. Manual annotation for image segmentation can be time-consuming. We try to keep the number of annotated samples as low as possible while still ensuring a reasonable variety in the data to learn robust models.

We annotate all samples in the training set for the segmentation of views, object views, and textfields. The difference between views and object views lies in their coverage and can be seen in the annotations depicted in Figure 3.2. Specifically, an object view focuses solely on the depicted part itself in a particular orientation, whereas a view covers a larger area that includes at least one object view along with other elements such as dimension lines, arrowheads, and numbers that are associated with the contained object views. Boundaries between and around views are sometimes vague, so it can occur that a single view actually contains multiple object views. An example of such a case is shown in Figure 3.3. Textfields are mostly rectangular tabular structures and are typically positioned in the bottom right of an ED.

Annotating smaller structures, such as arrowheads, dimension numbers, or surface symbols, requires more effort and time, which is why we annotate these objects in only around 50 EDs. In total, we annotated 1962 arrowheads, 950 dimension numbers, and 98 surface symbols. We combine arrowheads, dimension numbers, and surface symbol annotations to form the ground truth for the respective segmentation stage in our pipeline. Figure 3.4 shows an example of combined arrowhead, number, and symbol annotations.

Data Augmentation Data augmentation allows us to artificially increase the number of samples in the limited training data set and has been shown to be effective in increasing the robustness of DL models [ZCK+22]. The method uses the available data and applies different image processing techniques to the original samples. Each augmentation is applied with a certain probability and the augmentation parameters are typically drawn

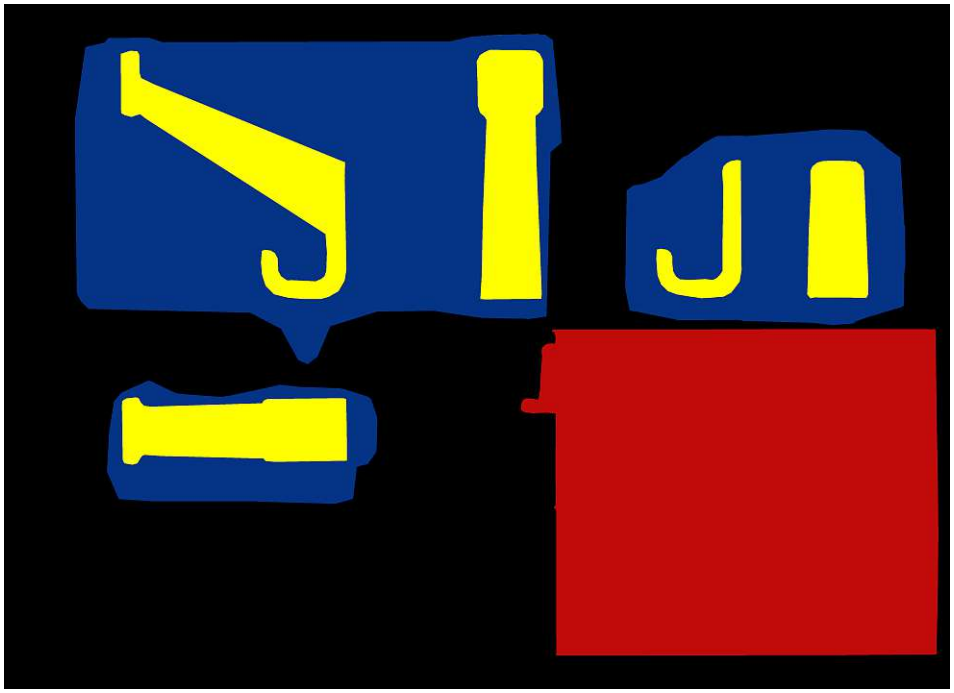


Figure 3.2: Annotation of textfields (red), views (blue), and object views (yellow).

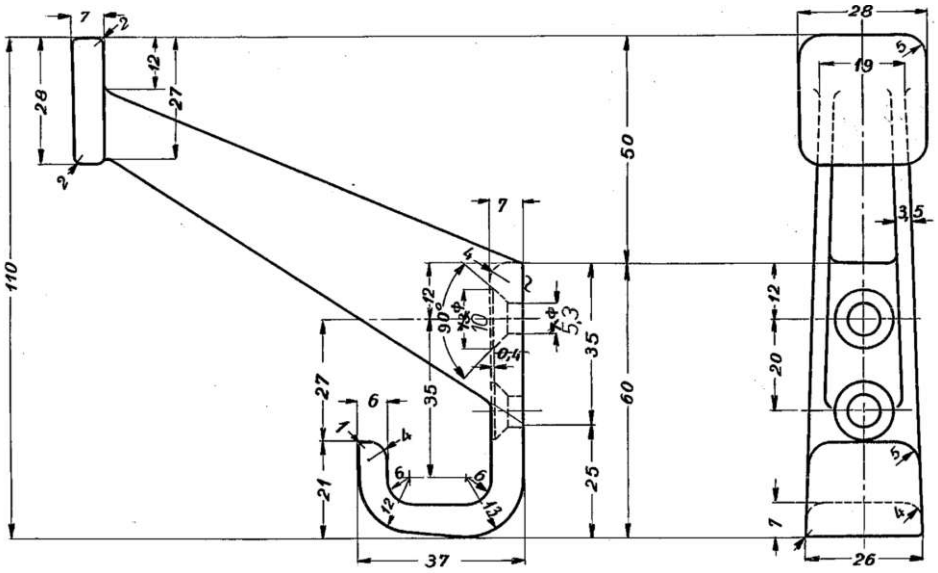


Figure 3.3: Unclear boundary between two object views. It is also not obvious which dimension lines belong to which object view. Image source Wiener Linien [wl].

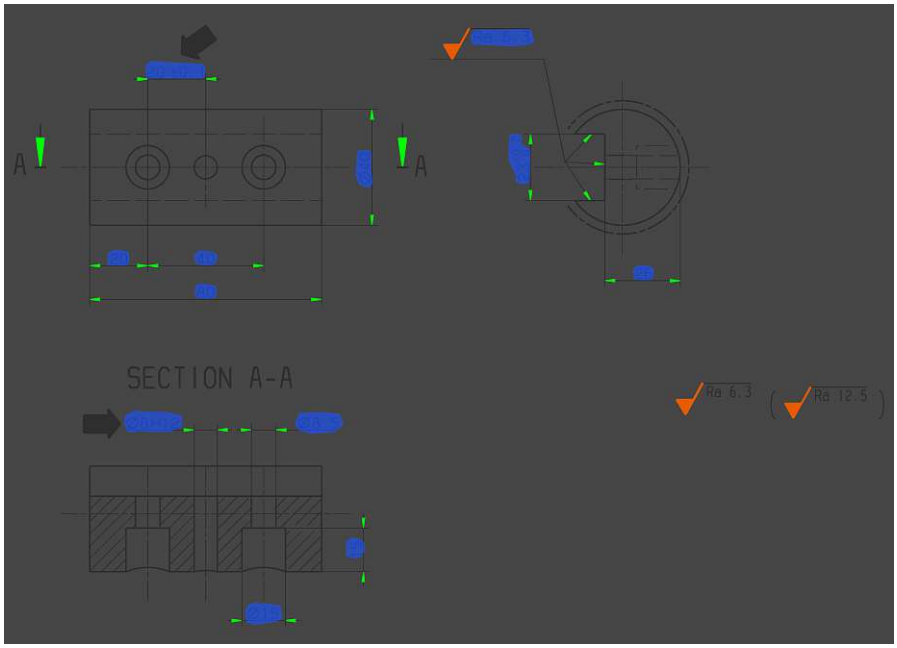


Figure 3.4: Annotation of arrowheads (green), dimension numbers (blue), and surface finish symbols (orange). ED source Wiener Linien [wl].

randomly from a range of allowed values. Every training sample during model optimization is processed with this sequence of randomized data augmentations. Therefore, the actual variety and number of different samples seen by the model in training is far greater than that present in the raw training data set. To enhance the diversity of our data set, we utilize various common image augmentation techniques. The augmentation pipelines differ for each training task, and we will provide a detailed explanation of them in the corresponding segmentation section along with examples.

Synthetic Data Synthetic data generation is used by many current DL computer vision methods to overcome the limitations of small data sets, for example in the method proposed by Schlagenhauf et al. [SNH23]. Examples of their generated synthetic data are shown in Figure 1.7. It has been shown to be effective in training segmentation models, for example, in the context of autonomous driving [SHL+24]. EDs typically follow strict rules with respect to the arrangement of their elements. We exploit this property of the underlying data to generate synthetic examples to increase the size of the training data set and to amplify the presence of less represented characteristics in the training data.

To improve the generalizability of the object view and view segmentation models, we generate synthetic data based on the views in the original training data set. We extract the views from all available training samples by cropping the respective regions from the original drawings based on the view annotation masks. Next, we create a blank image with a resolution of 2048x2048 pixels or randomly choose an existing image from the training data set as the starting point for our synthetic composite. If an image from

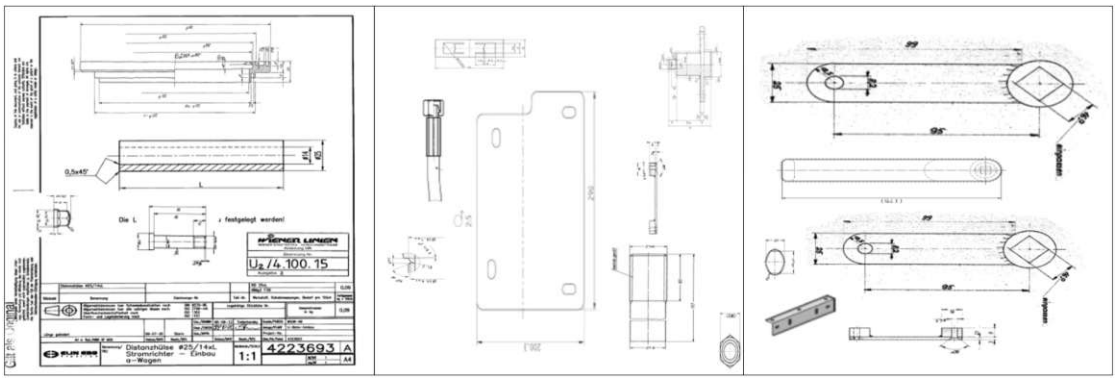


Figure 3.5: Synthetic data generated for training the view and object view segmentation models. ED source Wiener Linien [wl].

the training set is chosen, we remove all views from it, leaving only the textfields and the outer boundary. Subsequently, zero to ten views are randomly selected from the set of all views to be placed at random positions on the composite image. In order to introduce greater variability and mimic a wide range of potential real-world scenarios, we apply a series of transformations to each selected view. Specifically, they are subjected to non-uniform scaling, rotation, and adjustments in brightness and contrast with random parameters. A critical aspect of this strategy is the spatial arrangement of the views in the composite image. It is important that the placed object views do not overlap each other to avoid confusion during training. The annotation masks serve as a spatial guide for placing new views onto the composite image. Furthermore, the views must not overlap any textfields. This can be done by checking if the intersection of the textfield annotation mask of the current composite with the view annotation mask of the view to be placed is empty. Finally, we obtain a composite image with zero to ten views as well as annotation masks for textfields, views, and object views. We generate 2500 synthetic images to be added to the data set for training the object view segmentation model. Some examples of the resulting composite images are shown in Figure 3.5.

We develop an additional synthetic data generation scheme that takes advantage of the inherent rules of dimension lines in EDs for training an arrowhead segmentation model. Dimension lines are characterized by their start and end points marked by arrowheads and, in many cases, a line that connects them. Again, we start with a blank canvas to sequentially create a composite image containing multiple synthetically generated dimension lines. In some cases, we do not want a blank background, so we add random crops of the training samples that do not contain arrowheads. We develop an arrowhead generator that is capable of producing a wide variety of arrowheads and the corresponding annotation masks. This generator either selects a random sample from all annotated arrowheads in the training data or draws a simple black triangular shape. After that, non-uniform scaling is applied to the initial arrowhead. The generated arrowhead is then subjected to further random image deterioration procedures, including truncation,

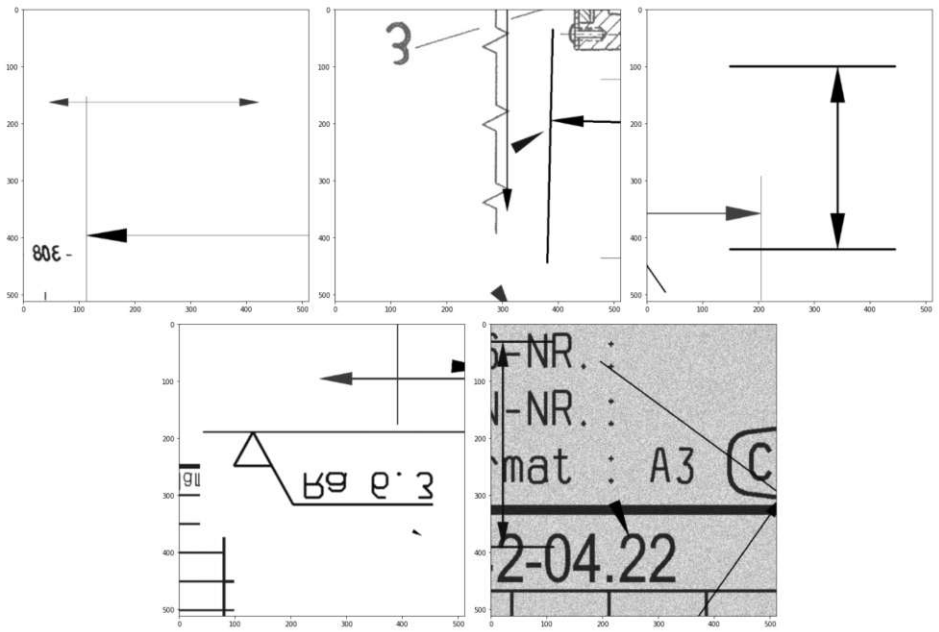


Figure 3.6: Synthetic dimension line data for improving the arrowhead segmentation model.

introduction of noise, contrast reduction, brightness adjustment, blurring, and deletion of random pixels. These manipulations aim to mimic the imperfections found in real-world EDs, thus improving the robustness of the segmentation model. Next, a dimension line generator is introduced. It chooses two random points in the image to form the start and end points of the dimension line and calculates the orientation between the points. Using the arrowhead generator, two arrowheads are created and placed at the start and end points. The generated arrowheads are rotated to align with the orientation of the dimension line. The second arrowhead is rotated an additional 180 degrees to ensure that it points in the opposite direction as the first arrowhead. To introduce further diversity, the direction of some arrowheads is perturbed by another small random rotation of ± 20 degrees. The dimension line generator then draws a line between the arrowheads with a probability of 75 % since we have encountered many examples of missing lines in real-world drawings. We randomly place between one and thirty generated dimension lines on the composite image. Finally, we apply the image deterioration strategies we mentioned above, but this time globally on the entire composite image. This yields examples ranging from low to high complexity with a large variety of image quality and enables us to increase the size of the data set in a scalable and controlled way. We generate 150 synthetic examples to be added to our data set for arrowhead segmentation training. Some crops of synthetic dimension line data are shown in Figure 3.6. We are aware that we could improve the synthetic data generators [SNH23], for example, by scripting a CAD tool, however, this is beyond the scope of this thesis.

3.3 Entity Segmentation

An integral part of our ED pipe is the semantic segmentation of textfields, views, arrowheads, dimension numbers, and object views, as these are the basic building blocks from which we derive dimension lines in subsequent processing steps. Image segmentation algorithms divide the input image into meaningful regions that can then be used for further processing. Semantic segmentation plays a critical role in many visual understanding tasks and is often the first step to derive complex information from images. It has been a key research topic for a long time and has applications in a multitude of fields, such as medical image analysis and autonomous vehicles. The task of semantic segmentation can be conceptualized as a pixel-wise classification problem [MBP+22].

In recent years, the methods have undergone a radical change with the introduction of DL-based segmentation approaches. These models have been shown to produce impressive results on popular benchmark data sets and outperform many of the classical methods. The most widely used architectures are based on convolutional neural networks (CNN). A breakthrough model is the U-Net, which was introduced in 2015 by Ronneberger et al. and was developed for medical image segmentation [RFB15]. A lot of the recent segmentation architectures are based on U-Net variants and follow a similar encoder-decoder scheme [MBP+22].

A further improvement was the proposal of the feature pyramid network (FPN) by Lin et al. [LDG+17]. This network architecture exploits the multi-scale hierarchical nature of CNNs to create feature pyramids with little extra cost, extracting information from the input image at multiple scales. An FPN consists of two main parts, which the authors call bottom-up pathway and top-down pathway. In addition, they use lateral connections to enable a flow of features from the respective scale in the bottom-up pathway to the top-down pathway. An FPN accepts common image encoder networks as the backbone to be used for bottom-up computation. In their paper, the authors employ a Res-Net101, however, this backbone can easily be replaced with other encoders as long as they follow a similar downscaling and pooling scheme. The top-down pathway upscales the feature maps multiple times, which could lead to hallucinations. This problem is alleviated by feature flow through the respective lateral connections at all scale levels of the FPN, with each lateral connection merging the feature maps of the same size from the two pathways. The architecture of the FPN proposed by Lin et al. is shown in Figure 3.7

We use FPNs for all segmentation tasks in the ED pipe, see Figure 3.1 (b) and (e), which segment the following entities:

- Segmentation of textfields
- Segmentation of views
- Segmentation of arrowheads, dimension numbers, and surface finish symbols
- Segmentation of object views (fine object borders)

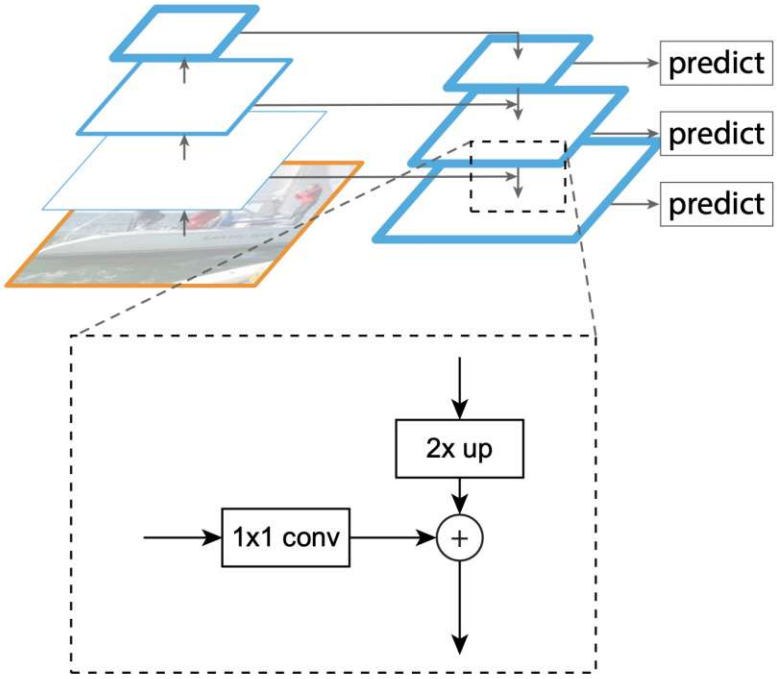


Figure 3.7: The FPN architecture proposed by Lin et al. [LDG+17]. The bottom-up pathway (encoder) is shown on the left and the top-down (decoder) is shown on the right side. The lateral connections can be seen between the two pathways.

Instead of a CNN like Res-Net, we employ a type of transformer model as the FPN backbone, namely the mix transformer encoder (MiT) proposed by Xie et al. in their Segformer architecture [XWY+21]. This encoder produces hierarchical multi-scale feature maps and can be used as the bottom-down pathway in the FPN. Image transformers apply the transformer architecture, originally designed for text processing, to visual data. They use an attention mechanism that maps a query and a set of key-value pairs to an output vector [VSP+17] which allows them to capture complex dependencies within an image. This is achieved by treating patches of an image as sequence elements, similar to words in a sentence, allowing the model to assess the importance of different areas of an image dynamically. This method contrasts with CNNs by not being constrained to local pixel neighborhoods, thus enabling a more global understanding of the image context. The attention mechanism is formulated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.1}$$

where (Q, K, V) are the query, key, and value, and d_k is the dimension of the query and key vectors. In typical transformers, multi-head attention is utilized, which consists of multiple attention layers that run in parallel, as shown in Figure 3.8.

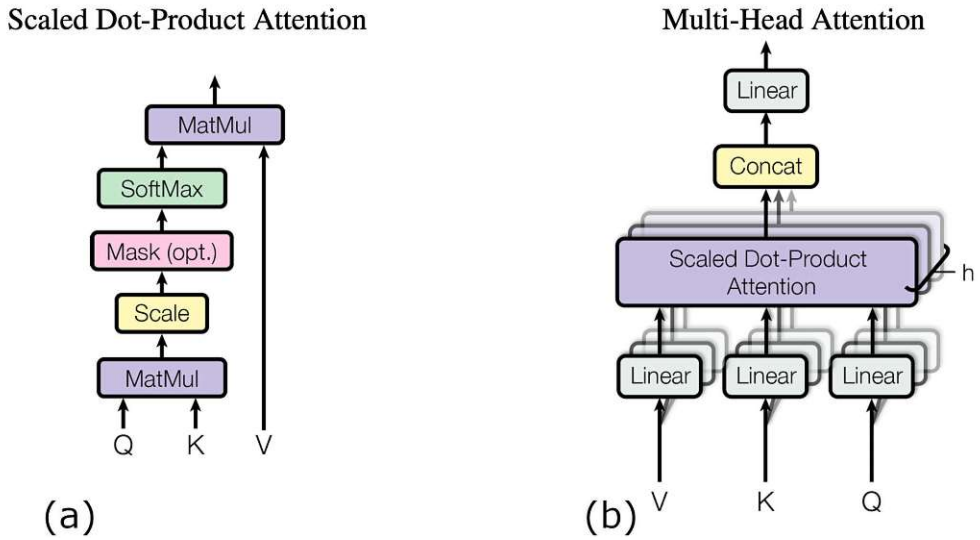


Figure 3.8: (a) Single attention mechanism. (b) Running several attention layers in parallel is referred to as multi-head attention [VSP+17].

Transformers have been shown to be effective in many problem domains, such as computer vision [HWC+23] and language processing [VSP+17], as demonstrated by the recent abundance of large language models. Due to their strong representation capabilities, transformer encoders are interesting options for use as the backbone in computer vision models [HWC+23], such as the FPN. A potential disadvantage of using transformers is their computational complexity of $O(n^2 * d)$ where n is the sequence length and d is the dimensionality of each head. The MiT by Xie et al. [XWY+21] employs a more efficient attention mechanism, reducing the complexity to $O(\frac{n^2}{R})$, where R is a reduction ratio. They set a different reduction ratio for each hierarchical stage. The size of the input to each subsequent stage decreases, so they set R to [64, 16, 4, 1]. The MiT divides an input image of size $H \times W \times 3$ into patches of size 4×4 pixels. These fine-grained patches are favorable for segmentation tasks according to the authors, as opposed to the 16×16 patches used by the original vision transformer (ViT).

The segmentation models in the pipeline are based on an FPN with an MiT backbone and differ only in their number of channels in the final output. Specifically, we use an FPN of depth five and the MiT-B5 encoder consisting of 81.4 million parameters. The encoder is pretrained on ImageNet1k [RDS+15] which should provide us with decent initial weights for training.

Textfield Segmentation Textfields are important parts of EDs and are of particular interest to our research partners for AM potential analysis. They hold valuable information about the parts depicted, for example, details about the material properties. Therefore, robust textfield segmentation is necessary and serves two purposes. First, we can identify and export crops of the input drawing that solely contain textfields, helping our partners

to process only relevant parts of the ED. Second, it plays a role in simplifying the input EDs for later processing stages in the pipeline. We noticed in Chapter 2 that textfields can cover significant areas of a drawing. A robust textfield segmentation allows us to confidently remove these regions from the input image for subsequent segmentation and processing stages, which should lead to less confusion and more stable results despite limited training data.

We want to perform binary segmentation of textfields, therefore, the textfield annotations serve as the targets for training, where zero-valued pixels in the annotation mask correspond to the background, and pixels with a value of one represent the textfield class. The input image is converted to grayscale and the pixel values are normalized to the range $[0, 1]$. The segmentation model benefits from global context, which means that it should predict the output by looking at the input image as a whole. The input EDs can have an arbitrary size and are often high-resolution scans with more than 100 million pixels, which is computationally prohibitive. Rescaling all training images and targets to a size of 512×512 pixels allows us to process an entire image in a single forward pass and fully exploit the characteristics of the MiT-FPN model. Such a heavy reduction in size is possible because textfields are relatively large structures, so we still retain all necessary information after downscaling. A data augmentation pipeline produces randomized samples during training and consists of the following processing stages:

- Horizontal flip, $p = 0.5$
- Vertical flip, $p = 0.5$
- Brightness adjustment, $p = 0.05$
- Contrast adjustment, $p = 0.1$
- Gaussian noise, $p = 0.1$
- Affine transformations, $p = 0.5$, translation: $\pm 25\%$, scale: uniform in $[0.5, 1.5]$, rotation: $\pm 10°$
- Rotation in $90°$ increments, $p = 0.75$
- Gamma adjustment, $p = 0.25$

This sequence of randomized augmentations yields a wide variety of training examples.

For textfield segmentation, we build a model based on the MiT-FPN architecture mentioned above, with a single output channel to perform a binary segmentation. The MiT backbone is pretrained on the ImageNet1k [RDS+15] data set which consists of more than one million RGB images. This is why we need to have three input channels even though we convert all images to grayscale. The output should be one if a pixel is covered

by a textfield and zero otherwise. This is achieved by a final sigmoid activation function that maps the output pixel values to the range $(0, 1)$ and is formulated as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

where x represents the pixel value to be mapped. As we can see in Equation 3.2, $\sigma(x)$ is differentiable and can be interpreted as a cumulative distribution function.

For model optimization, we calculate the Dice loss, which is a widely used loss function for image segmentation. Dice loss is a differentiable and smooth function that makes use of the Dice similarity coefficient (DSC) and can be expressed as follows:

$$\text{DSC} = \frac{1}{C} \sum_{c=1}^{C} \frac{2 \sum_{i=1}^{N} p_{i,c} y_{i,c}}{\sum_{i=1}^{N} p_{i,c} + \sum_{i=1}^{N} y_{i,c}}, \tag{3.3}$$

$$\mathcal{L}_{\text{DSC}} = 1 - \text{DSC}, \tag{3.4}$$

where C is the number of classes, N is the number of pixels, $p_{i,c}$ are the predictions generated by the model, and $y_{i,c}$ are the target values. It is a measure of the spatial overlap between the prediction mask and the target mask. As a result, Dice loss is robust to class imbalance. In contrast, the widely used cross-entropy loss is not robust against class imbalance, which could lead to an under-representation of smaller objects in the loss [YRN+22].

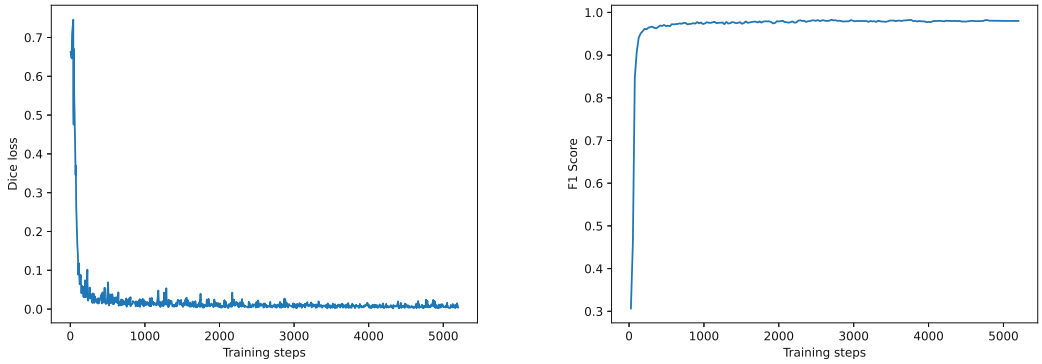
We calculate the F1 score during training for model validation, early stopping, and model selection. It is a measure of the harmonic average of recall and precision and combines these two values into a single metric [LQXL21]. Precision is the ratio of true positive (TP) predictions to the total number of positive predictions (TP + FP). TP refers to the number of positive instances that are correctly labeled as positive by the model. False positive (FP) refers to the number of negative instances that are incorrectly labeled as positive. Precision is an indication of how well the model performs in terms of correctly identifying positive instances. Recall is defined as the ratio of true positive instances to the sum of true positive and false negative (FN) instances. FN refers to the number of instances that are actually positive, but are incorrectly labeled as negative. Recall measures the effectiveness of the model to identify all relevant instances of a particular class. Precision, recall, and F1 score are calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.6}$$

$$F1Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.7}$$

We train the model for 200 epochs with validation after every epoch. Before each epoch, the training data are copied 20 times and shuffled. This means that every training sample



(a) (b)

Figure 3.9: Textfield segmentation model training. (a) Dice loss, (b) F1 score validation metric.

is seen 20 times by the model per epoch. Due to the online augmentation strategy, there will not be many duplicates. The chosen batch size is 32. We use the AdamW optimizer [LH17] with an initial learning rate (LR) of 10^{-4} and weight decay of 10^{-4}. Furthermore, we modulate the LR with two schedulers. In a warm-up period of three epochs, the LR is selected by $10^{i-4}, i = 0, 1, 2$, resulting in a ramp. After that, a second LR scheduler runs a cosine annealing scheme, gradually decreasing the LR until a minimum of 10^{-7} is reached. Figure 3.9 shows the training loss and validation metric of the training run of the selected model. The best validation result was achieved after 103 epochs and the trained model produces satisfactory results, which we discuss in detail in the results chapter.

View Segmentation An ED typically consists of one or multiple views that show the object in different 90-degree rotations and provide the viewer with the dimensions that characterize it. We define a view to contain not only the object itself, but in addition all linked dimension details, such as arrowheads, dimension numbers, and surface finish symbols. In contrast, we define an object view to contain only the region that covers the object in a view without additional entities outside of this area. Figure 3.10 shows an object view annotation superimposed on a view annotation to demonstrate this difference. Since the distance between the boundaries of two adjacent views can be small, it may happen that a view contains more than one object view. We do not want the view segmentation model to attend too strongly to fine-grained details, but rather to produce a rough, yet robust estimate of the relevant areas of the input image for further processing. View segmentation masks are used to remove all elements from the input ED that do not contain information that contributes to the finding of dimension lines. It is therefore a key component in our ED pipe to limit the search space in subsequent processing stages and to reduce the complexity of the drawing.

We train a model to produce binary segmentation masks of views in an input image and

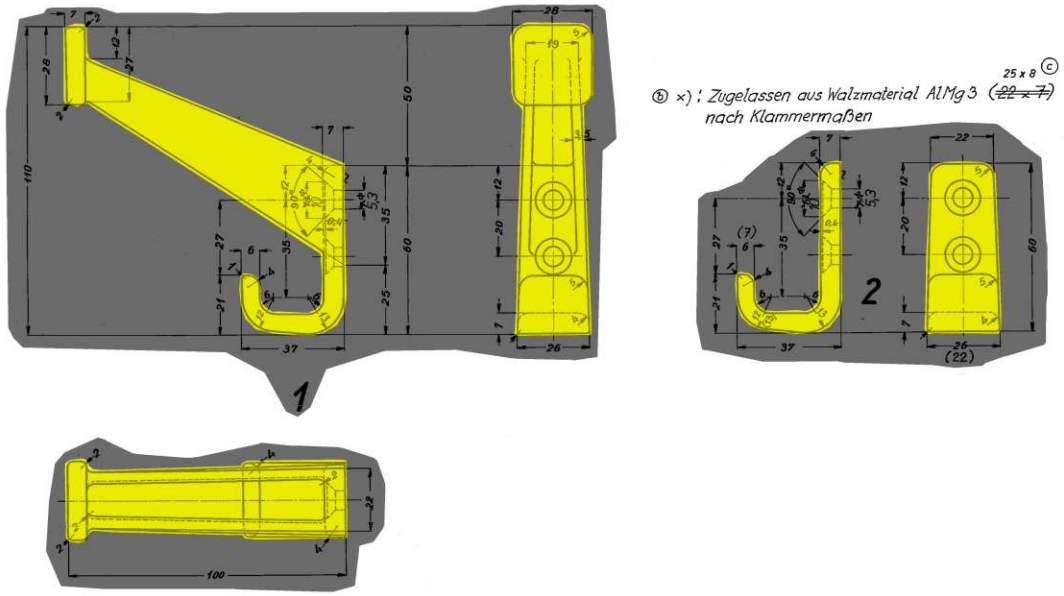
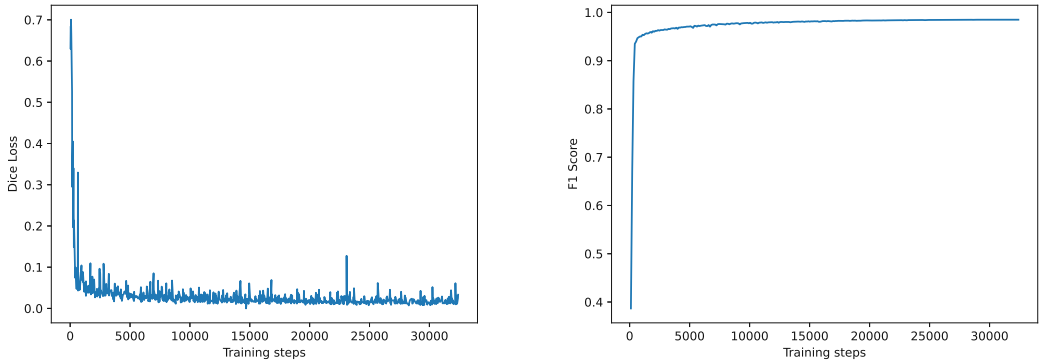


Figure 3.10: Difference between object view (yellow) and view annotation (gray). ED source Wiener Linien [wl].

use the same MiT-FPN model configuration as for textfield segmentation with an MiT encoder pretrained on Imagenet1k. Image preprocessing consists of rescaling the input to a fixed size of 512×512 pixels, conversion to grayscale, and normalization of the pixel values. The training data set contains 2500 synthetic samples that were generated as described above. We employ the same data augmentation strategy as described above for textfield segmentation and again calculate the Dice loss for optimization and the F1 score for validation purposes. We make use of the optimization setup of the textfield model and train the view segmentation model for 300 epochs with the LR scheduler introduced above. However, this time we do not increase the number of training samples by a factor of 20 because the 2500 samples are sufficient. The chosen batch size is 32, the initial LR is set to 10^{-4}, and weight decay is also set to 10^{-4}. Figure 3.11 shows the training loss and validation metric for the selected model. The best validation result was achieved after 294 epochs and the trained model produces satisfactory results, which we discuss in detail in the results chapter.

Object View Segmentation Finding the individual object views in an ED is critical for multiple reasons. First, it can be used to remove false dimension line estimates that connect two separate object views. Second, we can derive the object view grid structure to assign the relative rotation, and consequently the two corresponding principal axes to each object view. Third, a fine-grained object view segmentation with sharp boundaries can be used to directly infer the external dimensions of the part in the input drawing, which is an important fallback if the dimension line extraction fails or the external dimensions of the part in the drawing are not fully determined by its dimension lines.

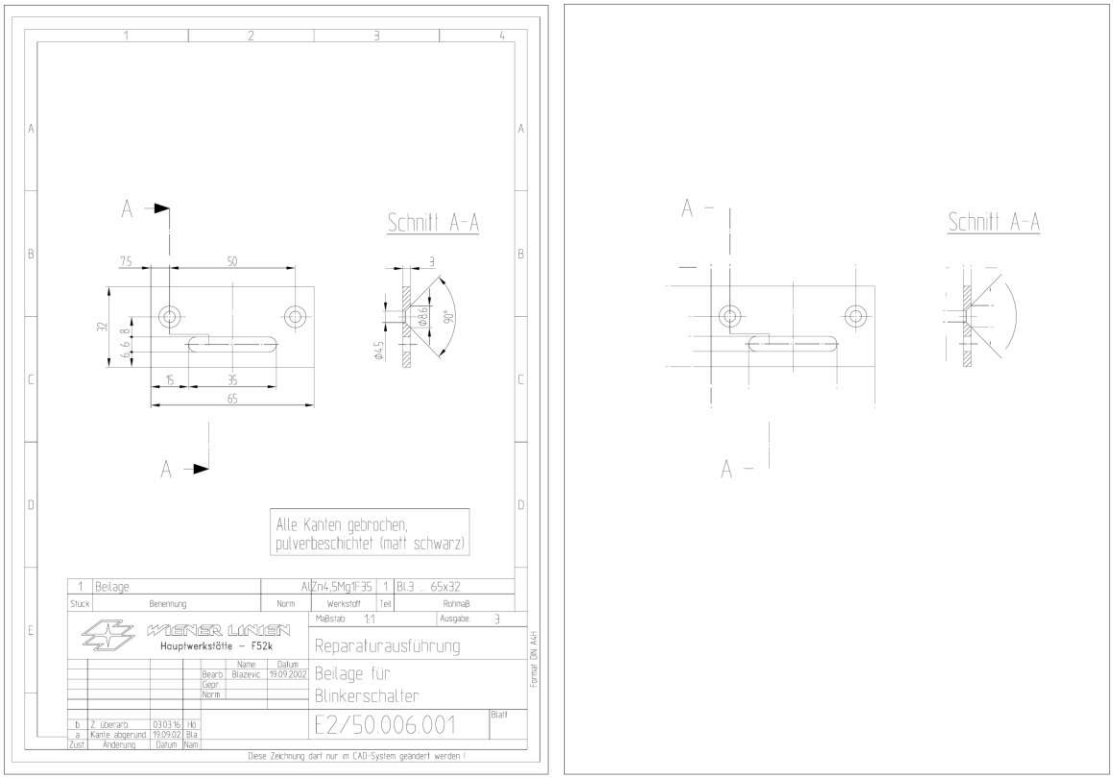


(a) (b)

Figure 3.11: View segmentation model training. (a) Dice loss, (b) F1 score validation metric.

Various characteristics of object views in EDs need to be considered to train a robust model capable of producing accurate segmentation masks. Object views can have an arbitrary shape and size, ranging from small structures that cover only a tiny portion of the input image to large structures that cover most of the drawing field. They might also contain holes or be very thin structures. The different semantic line types in EDs are typically drawn with different line thicknesses. However, in some cases this difference is not discernible and all line types have a similar thickness, making it challenging even for humans to determine the object view outline at first sight. In other cases, solid lines may be broken due to poor drawing quality, scan artifacts, or destructive preprocessing. Moreover, object views are typically organized in a grid, although such an arrangement is not obligatory. Some EDs show detail views or cuts of the part. These may look like regular object views, but are often not part of the main object view grid. Therefore, it is necessary to train a model that is invariant against scaling, translation, rotation, image quality, and object shape.

Object view segmentation is performed at a later stage in the pipeline after all other segmentation tasks and following an initial estimation of dimension line candidates, which will be described in the next section. We use this collected information to produce a simplified version of the input image that will then be processed by the object view segmentation model. Based on the view segmentation mask, we only keep the areas of the image that are covered by views, effectively removing many distractions such as textfields or outer boundaries. Next, we remove all segmented arrowheads, dimension numbers, and surface finish symbols. Finally, the initial dimension line estimates are used to delete lines that connect pairs of arrowheads in the input image by drawing a line with the same color as the background between the start and end points of each dimension line candidate. This first estimation will typically contain numerous false positives, for example, dimension lines connecting arrowheads that should be linked to



(a) (b)

Figure 3.12: (a) Original drawing; (b) Simplified version for object view segmentation. ED source Wiener Linien [wl].

two separate object views. However, since we are mostly interested in preserving the object view outlines, false positives do not pose major problems even if some of them may intersect an object view. After removing dimension line candidates, the resulting image mostly contains object outlines, which should help the segmentation model to deliver improved predictions. Figure 3.12 shows an unaltered input image in (a) and the simplified version in (b). It is clearly visible that a lot of clutter has been removed, which should lead to less confusing input data for the object view segmentation model. Although this simplification has benefits, we also prepare another version in which the arrowheads, dimension numbers, and dimension lines are still intact, as this will provide additional context to the model. In many cases, dimension lines can help to determine the actual extent of an object, in particular if the object view outlines are not clearly visible.

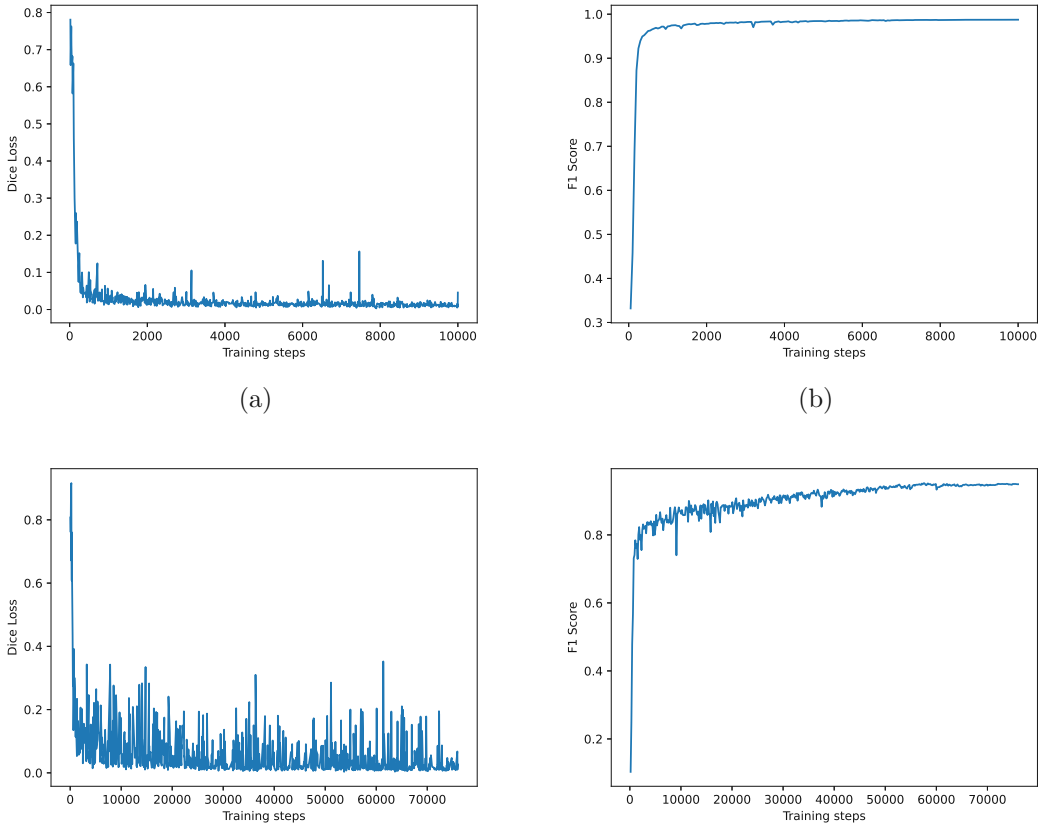
Our training data set consists of the original 110 drawings and 1500 synthetic samples generated with the object view data generator as described in Section 3.2. To achieve a slightly more realistic data distribution, each of the original samples is present twice in the training data set, resulting in a total of 1720 examples. All training images are

rescaled to a fixed size of 512×512 pixels and normalized to the range $[0, 1]$. We use the following data augmentation pipeline during training to increase the variety of examples even more.

- Horizontal flip, $p = 0.5$
- Vertical flip, $p = 0.5$
- Brightness adjustment, $p = 0.1$
- Contrast adjustment, $p = 0.1$
- Gaussian noise, $p = 0.1$
- Affine transformations, $p = 0.6$, translation: $\pm 35\%$, scale: uniform in $[0.5, 1.5]$, rotation: $\pm 20°$
- Rotation in $90°$ increments, $p = 0.75$
- Gamma adjustment, $p = 0.05$

We use the same MiT-FPN model architecture as for performing view segmentation. Dice loss and F1 score are calculated during training for optimization and validation purposes. We train two models with the AdamW optimizer and the same LR scheduler as used before in view segmentation. The first model is trained on simplified images for 100 epochs with a batch size of 64, while the other model is trained on the version of the data that still contains arrowheads, dimension numbers, and dimension lines for 200 epochs with a batch size of 16. The best validation results are obtained at the end of each training run, suggesting that the models do not show notable overfitting to the data, which we attribute to the wide variety of different examples that we generate. Figure 3.13 shows the training loss and validation metrics for the selected models. Figure 3.13 (a,b) shows the first model that is trained on heavily simplified drawings only. The training loss and validation metric curves look more jagged for the second model (c,d) that is trained on images with arrowheads, numbers, and dimension lines still present. We suspect that this greater uncertainty is mainly caused by the dimension lines which might be mistaken for object view contours.

Several steps are performed to obtain the final object view segmentation result. We predict masks for multiple rescaled versions of the simplified inputs with both models to boost scale invariance. In addition, we predict the masks on overlapping patches of the rescaled inputs, which are then stitched together. We are effectively using the individual models as ensembles when the overlapping patches are combined by averaging. Using different resolutions and crop sizes leads the models to attend to structures in the input image that may be missed when predicting on a single-scale input only. We choose the following pairs of rescaled longest side length and crop size:



(a) (b)

(c) (d)

Figure 3.13: Object view segmentation training of two models. (a)(b) model trained on heavily simplified training examples; (c)(d) model trained on simplified training examples, but arrowheads, dimension numbers, and dimension lines are still present. (a)(c) Dice loss; (b F1 score validation metric.

- 1024 pixel; 704 x 704 pixel
- 736 pixel; 512 x 512 pixel
- 704 pixel; 416 x 416 pixel
- 576 pixel; 384 x 384 pixel

The predicted stitched masks from both models are then merged by averaging. Figure 3.14 shows an example of the combined predictions of both models working as an ensemble, as well as predictions for individual scales. It is noticeable that the models have difficulties with some scales, but the combined predictions are able to produce a relatively accurate result. The masks are binarized by employing adaptive Gaussian thresholding, which

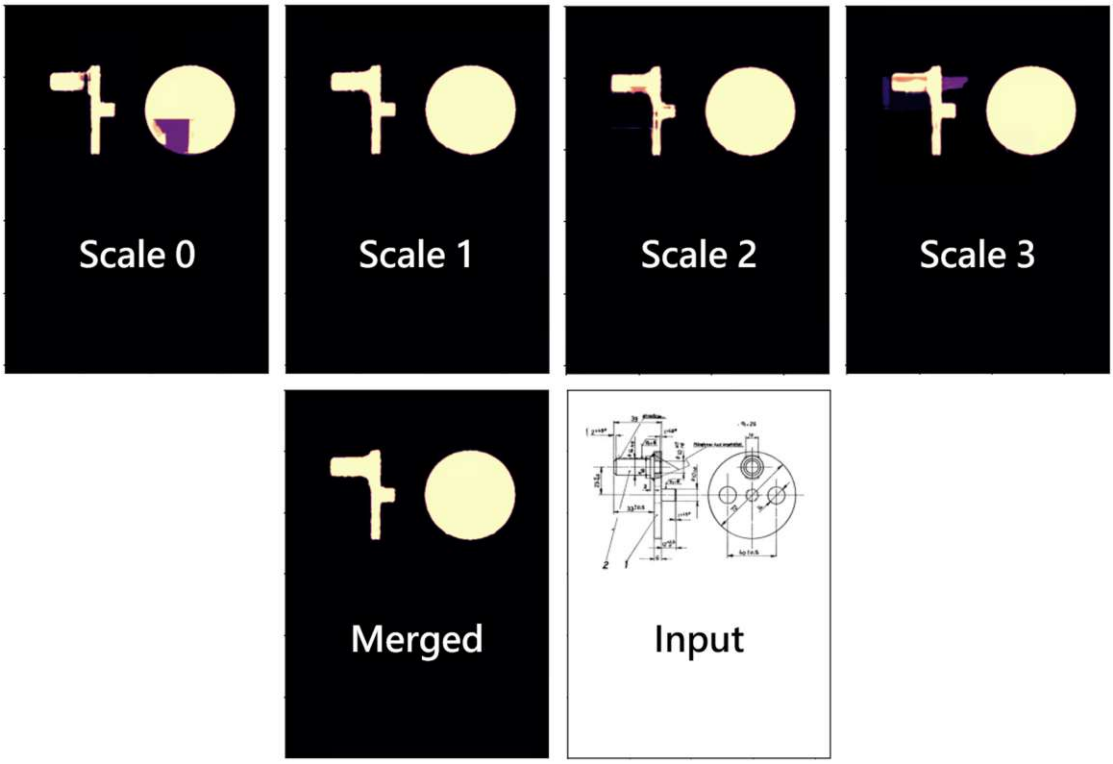


Figure 3.14: We merge predictions of differently scaled input images and crops to obtain the final object view segmentation result. ED source Wiener Linien [wl].

calculates the threshold values on windows of the predicted masks and is not as aggressive as a global threshold. Finally, we run morphological closing with a kernel size of 9×9 pixels before rescaling the masks back to the size of the original input image.

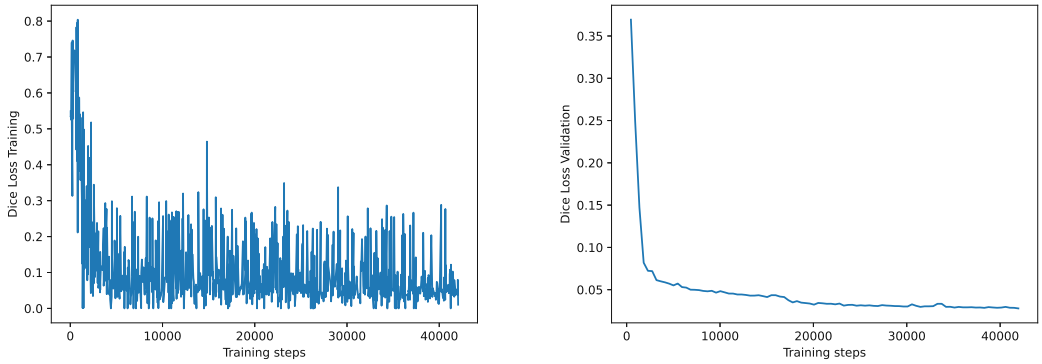
Arrowhead, Dimension Number, and Surface Finish Symbol Segmentation Arrowheads and dimension numbers are key components of EDs and provide the viewer with the dimensioning characteristics of the depicted part. Detecting them is critical to successfully extract the external dimensions from a drawing. Furthermore, in Chapter 2, we discussed that arrowheads and their respective orientations hold a lot of information about a part and its complexity. Arrowheads are one of the most reliable features of EDs throughout time, due to their distinctive shape and their presence in most drawings. They are the basis of our dimension line reconstruction approach that we describe in the Section 3.4. Therefore, it is critical to train a robust segmentation algorithm which is capable of detecting arrowheads in drawings of different time periods, styles, and resolutions. The resulting masks must be accurate enough to determine the arrowhead orientation. This is why we opt for segmentation instead of object detection, which would result in axis-aligned bounding boxes. Arrowheads are rarely perfectly vertically or horizontally oriented in real-world drawings. The image might be scanned off-axis, while hand-drawn examples are typically not precise enough, leading to skewed arrowheads.

The shape of arrowheads in EDs shows a wide variety, ranging from long and thin structures to arrowheads of almost equal length and width. The latter case would always result in square bounding boxes, making it impossible to infer orientation. High-fidelity segmentation masks, on the other hand, can be used for orientation estimation, and additionally it is simple to derive bounding boxes from them. In most cases, arrowheads, dimension numbers, and surface finish symbols are comparable in size and do not overlap, which is why we choose to segment them in one pass with the same model. Processing arrowheads together with dimension numbers gives the model additional context since these entities typically follow strict rules in how they have to be positioned relative to each other.

We use the training dataset consisting of 50 fully annotated drawings together with 150 synthetically generated samples. The model should produce accurate segmentation masks, so we have to predict on patches of the input image. For this reason, we resize each image in the training set so that the longest edge is 3072 pixels and then randomly sample 15344 crops of size 512×512 pixels. Most of these crops should contain arrowheads or dimension numbers. We achieve this by randomly sampling an image and checking if the corresponding crop of the annotation contains a certain number of pixels belonging to the respective class. The 15344 crops contain 3000 empty samples, meaning that there are no arrowheads, dimension numbers, or surface finish symbols present, but other parts of the drawing are preserved, for example object view outlines. 9000 of the other samples contain at least one arrowhead and sometimes dimension numbers and symbols. The rest do not contain arrowheads but rather numbers and/or symbols. There is a limited number of surface finish symbols in the original drawings, so we randomly insert synthetic symbols when sampling the crops to enhance their significance in the overall class distribution. Again, we perform online data augmentation with the same augmentation pipeline as in object view segmentation, but with a larger range of random values for the brightness, contrast, and transformation parameters.

We use the same MiT-FPN model architecture, but this time with four output channels, producing masks for the background, arrowheads, dimension numbers, and surface finish symbols, including welding symbols. The training setup is similar to before with the use of the same cosine annealing LR scheduler. We train the model for 100 epochs with multilabel Dice loss for optimization and validation. The best validation result is achieved after 91 epochs and the training loss and validation metric is shown in Figure 3.15. The plots suggest that no substantial overfitting occurs. The jagged appearance of the training loss plot stems from the large variety of examples and heavy data augmentation strategy, ensuring that the model does not often see the exact same image multiple times during optimization.

We split the input image into overlapping patches of size 512×512 pixels for inference and combine the resulting masks by averaging them to produce the final prediction. The model produces satisfactory outputs, which we will discuss in detail in the results and evaluation chapter.



(a) (b)

Figure 3.15: Arrowhead, dimension line, surface finish symbol segmentation model training. (a) Training Dice loss; (b) Validation Dice loss.

3.4 Dimension Line Processing

Dimension lines in an ED are critical for determining the external dimensions of the depicted part. We utilize the results of the entity segmentation to find potential dimension lines and to link them to corresponding dimension numbers. This is achieved in a multi-step process starting with an initial guess and subsequent refinement. Dimension lines are commonly oriented either horizontally or vertically, though there are cases where they may have a non-standard direction. Additionally, if the original drawing was scanned at an oblique angle, the resulting digital version may lack horizontal or vertical lines altogether. Therefore, we cannot depend solely on detecting horizontal or vertical lines, but instead, we must address scenarios with arbitrary directions.

Initial Estimation of Dimension Lines We start by estimating a set of potential dimension lines based on the arrowhead and view segmentation outputs. A typical dimension line consists of two arrowheads that point in opposite directions and are part of the same view. In this thesis, we are not concerned with finding angles measured in degrees or dimension sets containing only a single arrowhead where the assumption about arrowhead orientation does not hold. First, we perform connected component labeling on the segmentation results to obtain a set of arrowheads and a set of views, and link each arrowhead to the respective view that contains it. We perform principal component analysis (PCA) on the arrowhead segmentation masks to calculate the arrowhead orientations. Next, we find all matching pairs of arrowheads to form the initial dimension line estimate. Matching arrowheads must lie within the same view and their orientations must agree. This can be determined by calculating the absolute dot product between the two orientation vectors and checking if this value satisfies a tolerance:

$$|\boldsymbol{d_i} \cdot \boldsymbol{d_j}| \approx 1, \tag{3.8}$$

where $\boldsymbol{d_i}$, $\boldsymbol{d_j}$ are the normalized orientation vectors of the start and end arrowheads. However, this simple criterion is not sufficient on its own because EDs often contain parallel dimension lines where all arrowheads have similar orientations, which would lead to a large number of false positives, for example, lines crossing other lines. Therefore, we search for matching candidates in a narrow strip that extends linearly to $\pm\infty$ in the direction of the starting arrowhead. We construct two parallel lines in the same direction as the starting arrowhead at an orthogonal distance $\pm o$ from the arrowhead center and accept candidates only if they are positioned within this strip. It can be formulated as follows:

$$\boldsymbol{b}_{\pm} = \boldsymbol{p} \pm o\boldsymbol{d}_{\perp} \tag{3.9}$$

$$c(\boldsymbol{x}) = sign(\boldsymbol{d} \times (\boldsymbol{b}_{-} - \boldsymbol{x})) \neq sign(\boldsymbol{d} \times (\boldsymbol{b}_{+} - \boldsymbol{x})), \tag{3.10}$$

where $\boldsymbol{d}$ is the normalized arrowhead orientation, $\boldsymbol{d}_{\perp}$ is the normalized orthogonal direction, $\boldsymbol{p}$ is the arrowhead center, o is the offset that determines the width of the strip, $\boldsymbol{b}_{\pm}$ are the points that support the parallel boundary lines of the strip, $\boldsymbol{x}$ is the center of the arrowhead to be evaluated, and c is the criterion that must be fulfilled in order to accept this arrowhead as a matching candidate. The variable o is a hyperparameter that controls the tolerances, and we set $o = 0.1$. Such a large tolerance is not necessary for CAD generated drawings, but is important for many hand-drawn examples where the arrowhead orientations are typically unstable and show a large deviation from the actual direction of the respective dimension line. Due to this instability, we additionally evaluate the agreement of the two arrowhead directions with the direction between the two arrowhead centers $\boldsymbol{d_{ij}} = \frac{\boldsymbol{x}-\boldsymbol{p}}{||\boldsymbol{x}-\boldsymbol{p}||}$ and compute an overall matching score s_{ij} as follows:

$$s_{ij} = \begin{cases} \frac{|\boldsymbol{d_i}\cdot\boldsymbol{d_j}|+|\boldsymbol{d_i}\cdot\boldsymbol{d_{ij}}|+|\boldsymbol{d_j}\cdot\boldsymbol{d_{ij}}|}{3} & , i \neq j \\ 0 & , i = j \end{cases} \tag{3.11}$$

where the score is in the range $[0, 1]$ and zero if the start and end arrowheads are identical. We keep the match if $s_{ij} \geq s_{min}$, another hyperparameter that we set to $s_{min} = cos(12°)$. The initial set of dimension line estimates includes all matching pairs of arrowheads, with the exclusion of all duplicates that have swapped start and end points. This initial guess might still contain many false positives, but is sufficient for producing the simplified version of the input drawing for the object view segmentation as described in the previous section.

Refinement, Linking to Object Views, and Dimension Numbers We run a series of refinement operations on the initial set of dimension lines to further reduce the number of false positives. At this stage in the pipeline, we already obtained the object view segmentation masks, so we can link the arrowheads to the respective object views, which allows us to exclude dimension lines with arrowheads belonging to two different views. We employ a similar scheme to link arrowheads and object views as before when matching arrowheads. This time we extend a strip in the orthogonal direction of the arrowhead and check if it intersects an object view, as illustrated in Figure 3.16. It is possible that an arrowhead is linked to more than one object view, for example, if it lies in between

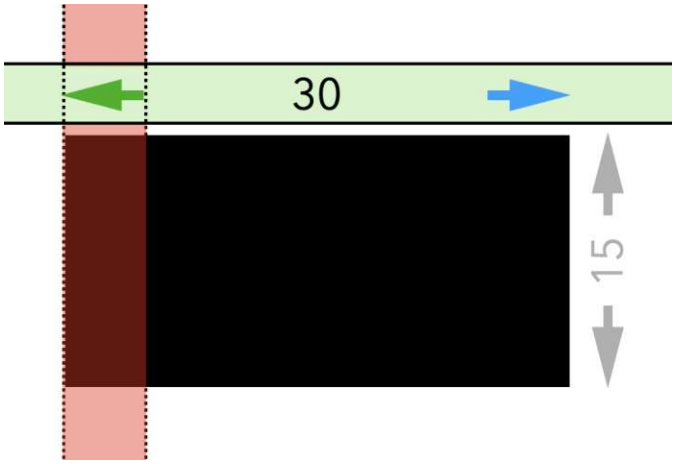


Figure 3.16: Linking the green arrowhead to the black object view and the blue arrowhead.

two object views, but these cases do not cause problems and will not yield false positives because of the orthogonality to the direction of the dimension line. After linking, we can remove false dimension lines by asserting that:

$$|A_{ov} \cap B_{ov}| = 0, \tag{3.12}$$

where A_{ov} and B_{ov} are sets containing the linked object views of the first and the second arrowhead of a potential dimension line respectively. Furthermore, we reject all dimension lines that cross more than one object view.

In most cases, an arrowhead is only part of one dimension line, except for chained measurements, which are of lower importance for determining the external dimensions of a part. Therefore, we split the current set of lines into two. The first set contains lines where each arrowhead is exclusively linked to only one line in the set, whereas the rest are placed in the second set for further refinement. There are multiple reasons why some arrowheads may be linked to more than one dimension line, including chained measurements, inaccuracies stemming from excessive tolerances, or errors in the object view segmentation masks. We proceed to remove outliers from the first set by applying a Hampel filter [PNAG16] to the vectorized scores s_{ij} and placing these outliers in the second set.

Next, we represent the second set as a graph G where the arrowheads are the vertices and the edges correspond to potential dimension lines connecting the arrowheads. This structure allows us to efficiently handle cases such as chained measurements or crossing lines. We divide G into several subgraphs H_i, where each subgraph contains a connected component of G. In each H_i we try to find clusters of edges that partially or completely cover other edges, which could indicate the presence of a chain measurement. Since we try to extract the external part dimensions, we are only interested in the largest possible extent of a chain measurement. This is obtained by adding the dimension numbers of all subsegments or, in some cases, choosing the largest dimension number. We iteratively

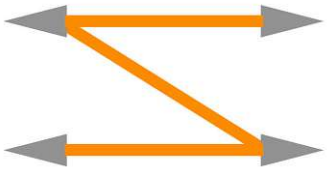


Figure 3.17: Abstract example of a Z-path.

determine the starting and end points of the chain measurement and insert a new edge that connects the two arrowheads into G. Then we remove all edges that are part of the chain and delete unconnected arrowheads from G. After that, we remove all valid dimension lines from G and put them in the first set. False positive dimension lines often introduce Z-paths, which are another type of line arrangement that we want to untangle. Figure 3.17 gives an example of a Z-path. They are similar to chains, but contain sharp changes of direction and typically consist of two or more valid edges that are connected by false positives. Z-paths are acyclic and each node has degree $d(v) \leq 2$ with the starting and end node having degree $d(v) = 1$. If a possible Z-path is detected, we perform a depth-first traversal of the subgraph H_i starting at a node of degree 1 and count the number of significant direction changes along the path. In a path of n edges, we need to count $n - 1$ direction changes to confirm that we have found a z path. We keep all edges with even indices on the path $(0, 2, 4, ...)$, move them to the first set of dimension lines, and discard edges with uneven indices. We observed that Z-paths often start with a valid dimension line, so this heuristic yields acceptable results in most scenarios. In the next step, we handle crossings of dimension lines. Valid lines must not cross each other, and therefore we should eliminate all crossing lines from both sets. If two lines intersect and are linked to four distinct arrowheads, we detected a potential crossing to remove. However, by doing so, we may remove valid lines that happen to be crossed by an invalid line. To alleviate this issue, we check if one of the crossing lines shares an arrowhead with a third line, and if that is the case, assume that it is invalid and remove it. Figure 3.18 depicts an initial set of dimension lines in (a) and the remaining set of lines after refinement in (b). As preparation for later pipeline stages, we cluster the remaining dimension lines by their overall orientation in vertical, horizontal, and diagonal lines. Additionally, we compute corrected endpoints of the dimension lines by finding the intersection points of the arrowhead tips with a line passing through the center of the arrowhead in the direction of the dimension line.

We still need to link the refined dimension lines to the respective dimension numbers, which are typically located at the half-way point between two arrowheads but, in some cases, can also be positioned outside the span of the dimension line. This again necessitates the use of tolerances in the search for potential dimension numbers, which could lead to multiple numbers being associated with one line and, conversely, one number being linked to multiple lines. Each dimension line is enclosed within a non-axis-aligned bounding

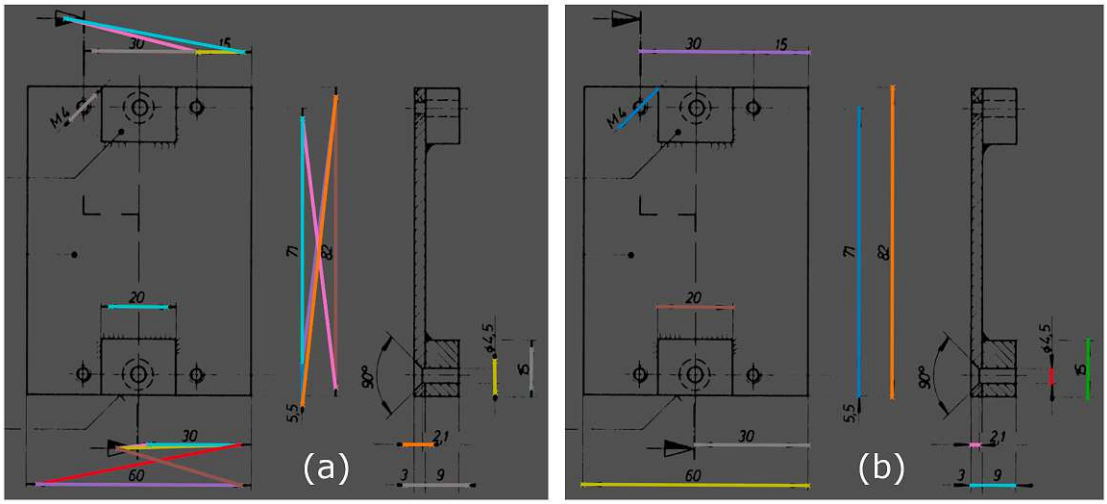


Figure 3.18: (a) First dimension line estimation; (b) filtered set of dimension lines on the right side. ED source Wiener Linien [wl].

box that extends a specific tolerance distance in the orthogonal direction of the line and in the line direction at both end points. Next, we intersect the bounding boxes of all dimension numbers with these newly generated boxes and assign the numbers to the respective dimension line if the intersection is not empty.

3.5 View Grid Structure

Object views in an ED are typically arranged in a grid structure. The transition from one grid cell to the next implies a 90° rotation. When transitioning horizontally, the object rotates around the vertical axis, whereas transitioning vertically results in a rotation around the horizontal axis. This concept is demonstrated in Figure 3.19. Special cases of object views typically drop out of the grid alignment, for example, detail views, certain cuts through the part, or unfolded representations. In this thesis, we process drawings that depict only a single part. It is important to mention that there exist EDs that show multiple parts in an assembly drawing, resulting in multiple grids. However, this is beyond the scope of this work. Determining the grid structure is essential to later assign the appropriate axis labels to the individual object views when computing the final external dimensions of the part. We make use of the results from previous pipeline stages to construct the grid, namely, the object view segmentation masks and the refined dimension lines. A valid grid has specific properties that we can exploit to reconstruct it. Due to the 90° rotations, one axis remains in place, thus, the length of the respective side of the part does not change. Figure 3.19 shows six object views in a valid grid structure. We observe that the height of the part is the same in all four horizontally aligned object views (C, A, D, F), while the width is constant in the three vertically aligned object views (B, A, E). Furthermore, dimension lines cannot extend beyond the height or width of each

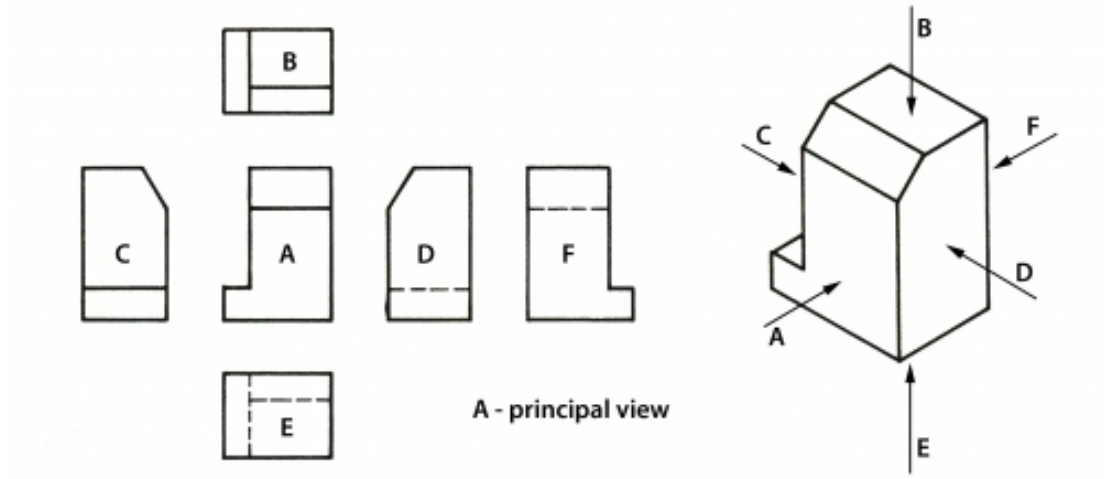


Figure 3.19: Views in an ED arranged in a grid structure. Transitioning between grid cells causes a 90° rotation of the depicted object. Only vertical and horizontal transitions are allowed. It is visible that the side length corresponding to the rotation axis remains constant, for example, the height of views C, A, D, and F is the same, while the widths of B, A, and E are equal. Image source: [mcgc]

individual object view. With this in mind, we construct a map in which we count the coverage of each pixel by object views or dimension lines. Figure 3.20 illustrates the steps required to build the view grid map. We first create masks for the coverage of horizontal and vertical dimension lines as is shown in 3.20 (b) and (c). Diagonal dimension lines contribute to both line coverage masks. Next, we fetch the fine object view segmentation mask (d) and the corresponding object bounding boxes (e). Finally, we add all masks to obtain the view grid map (f) where the grid structure is clearly visible when focusing on the regions of the map with the highest coverage count. A simple threshold operation yields the image regions that most likely belong to the object view grid. Some EDs show views that are not dimensioned completely, which would result in falsely rejected object views from the grid. To alleviate this problem, we use a similar map as before, but also add a binary version of the object view segmentation output where the object view class pixels have a value of one and the rest are filled with zeros. The object view segmentation masks may be fragmented such that a single object view actually consists of multiple smaller masks with nested bounding boxes. The merging of overlapping bounding boxes and applying these corrections to the map addresses this issue and ensures that each object view only contributes a single bounding box to the map.

After building the map, we can find the corresponding object views by intersecting them with the map. In addition, we generate new bounding boxes based on connected

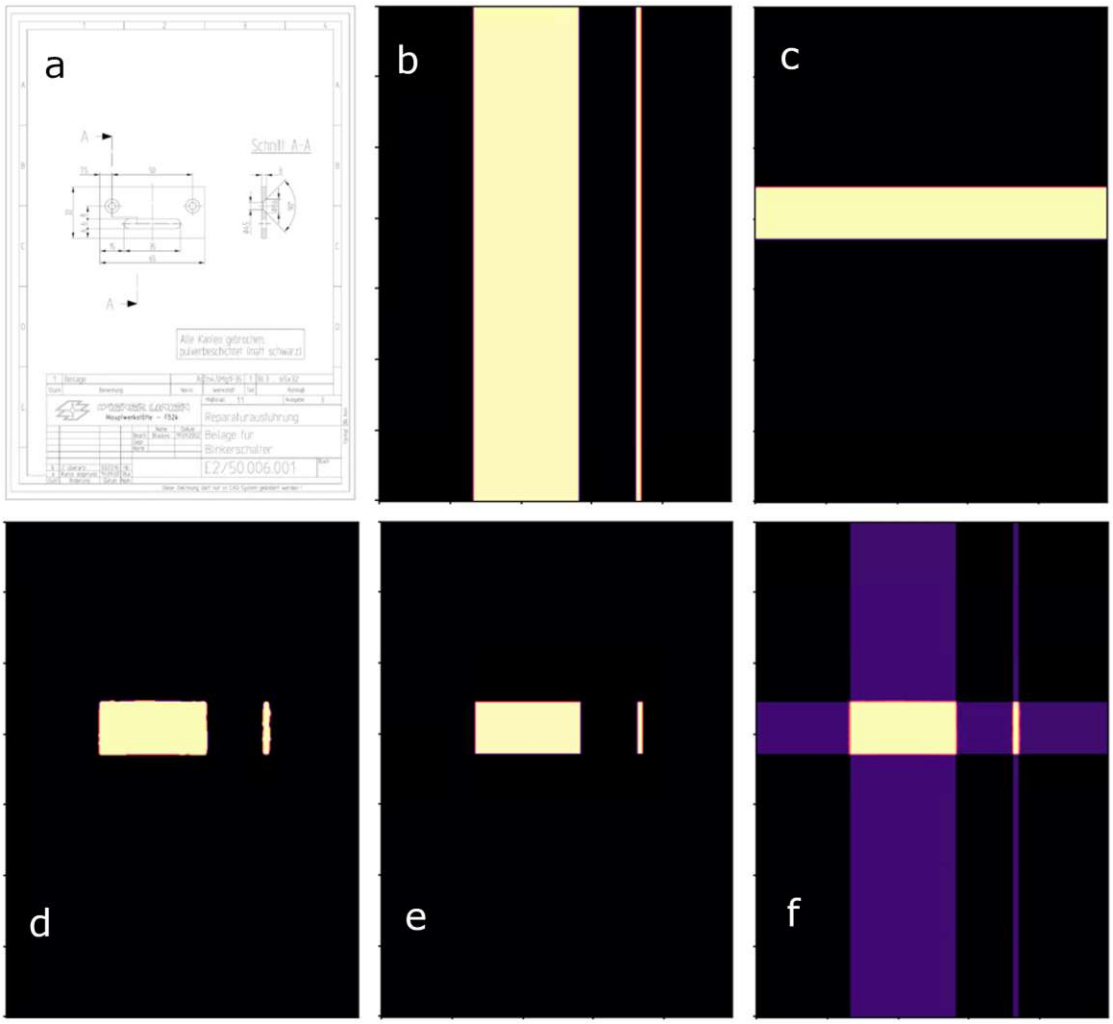


Figure 3.20: Construction of the view grid map. Input ED (a), horizontal dimension line coverage (b), vertical dimension line coverage (c), object view segmentation (d), object view bounding boxes (e), final grid map (f). ED source Wiener Linien [wl].

component labeling of the map and assign them to the respective object views. To reconstruct the grid structure, we compute two adjacency matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$, of size $n_{ov} \times n_{ov}$ where n_{ov} is the number of object views. The matrix $\boldsymbol{A}$ represents the horizontal adjacency, while the matrix $\boldsymbol{B}$ represents the vertical adjacency of object views on the grid.

For each pair of object views (v_i, v_j), we check the alignment of their bounding boxes along the horizontal axis by assessing their vertical boundary consistency within a specified tolerance. We identify the maximum height and derive a tolerance threshold as a percentage of this height. Subsequently, we examine whether the discrepancies between the minimum and maximum Y coordinates of both bounding boxes fall within this tolerance, indicating horizontal alignment. We set $a_{ij} = 1$ if horizontal alignment is detected and $a_{ij} = 0$ otherwise. Similarly, we assess the vertical alignment of bounding boxes by comparing their horizontal boundaries (X coordinates) within a tolerance and set $b_{ij} = 1$ if vertical alignment is detected. These connections are used to build a graph of the grid structure to track rotations between object views and to determine the correct axis labels.

3.6 OCR

Dimension numbers in EDs specify the spatial characteristics of the depicted objects. At this point in the ED pipe, we already know their location and potential association with the dimension lines. We employ OCR to convert the visual content into numerical data, allowing us to determine the length of the dimension lines true to scale. An OCR model based on MMOCR [KSL+21] is fine-tuned on additional synthetically generated custom data. The data generation and model training tasks were kindly carried out by Lisa Maria Kellner, a researcher at VRVis. In this section, we will give a coarse overview of the training data generation and training procedure as a detailed description of OCR is beyond the scope of this thesis.

Instead of running OCR on the entire image, we can exploit the gathered knowledge about the location and extent of the dimension numbers. This significantly simplifies the input to the OCR model by only processing snippets of the image that contain a single dimension number. In contrast, common OCR models, such as tesseract [tes], were easily confused when faced with a complete ED as input, leading to unreliable results. Furthermore, they were still unreliable when processing single-number snippets and seemed to yield good results only on continuous blocks of text. This has several reasons, for example, the radically different data distribution that was used to train these models compared to our specialized case of EDs. Another reason is the orientation of the dimension numbers. OCR models seemed to perform well on text that runs from left to right, but struggled if the text was at an angle, running vertically, or upside down. The vertical orientation in particular is a common case in EDs across all time periods. In addition, the models had problems with text at vastly different scales and often ignored small numbers. Many dimension numbers contain special characters, such as Ø, ~, or ±,

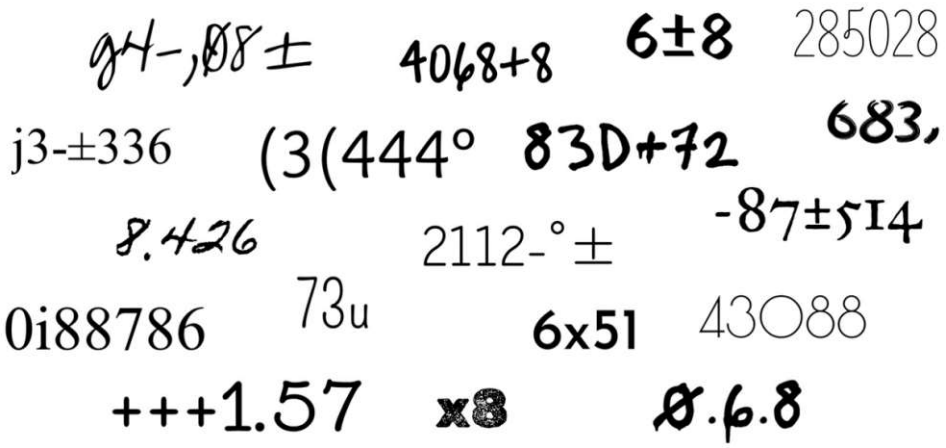


Figure 3.21: Synthetic dimension number examples to fine-tune the OCR recognition model. Image source Lisa Maria Kellner [vrv].

that are not even present in the vocabulary of standard models. These problems motivate the fine-tuning of a custom OCR model. As mentioned above, acquiring abundant and reliable annotated training data is a common problem when optimizing a model. Therefore, we generate a synthetic training data set that matches our purposes. We utilize a subset of Google Fonts [fon], a library of more than 1600 distinct fonts, including some that are similar to handwriting, and extend the standard vocabulary with the aforementioned special characters. Next, we generate short text sequences of varying length with random fonts at random scales and randomly sampled characters biased towards digits and special characters. This is useful because dimension numbers in real-world EDs often contain more than just digits and supports model generalization. Figure 3.21 shows some examples of synthetically generated examples that were used to fine-tune the OCR model. We do not want to lose the pretrained capabilities of the OCR model, which is why we add a subset of the original training data to the training set for a fifty-fifty split between synthetic and original samples. To overcome the problem that text typically flows from left to right, we process each snippet at four 90° rotations during model inference and store each prediction as a potential numerical value of the respective dimension number. In the next stage of the pipeline, we will determine which of these values is correct. This approach allows us to fine-tune the model on correctly oriented text that flows from left to right only, which should facilitate faster convergence of the training procedure.

3.7 External Dimensions and Pipeline Outputs

In the last stage of our ED pipe, we extract the external dimensions of the part depicted in the input drawing using the results generated in previous stages. We output dimensions in X, Y, and Z based on the detected dimension lines and, in addition, based on the

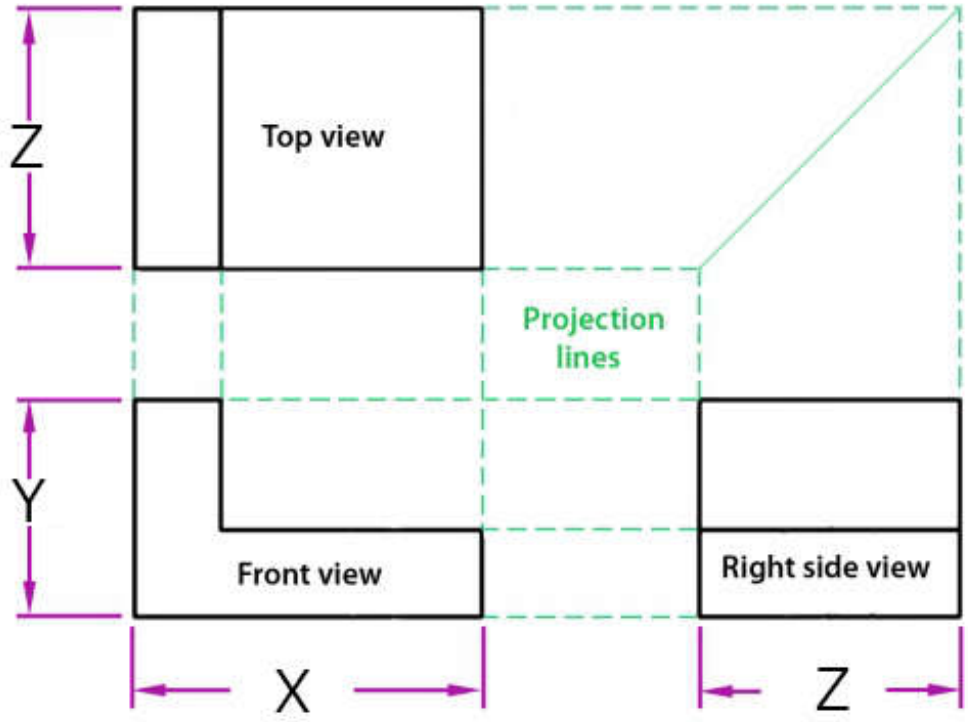


Figure 3.22: Three views arranged in a grid with assigned axis labels. Note that each view has two associated axes. Image source: [mcgb]

bounding boxes of the segmented object views because sometimes the extensions of the depicted part are not fully specified by the dimension lines alone. The results will be available measured in pixels and also in millimeters by using the OCR results and calculating the OCR-to-pixel ratio. This makes the system more robust to erroneous predictions from the OCR model and is necessary, since we need to determine the correct orientation of the dimension number texts, which we predicted in four different 90° rotations.

External Dimensions and OCR-to-Pixel Ratio We build a graph that represents the structure of the view grid. The nodes are the individual object views, and the edges stem from the horizontal and vertical adjacency matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. The resulting graph is acyclic and can be used to assign axis labels to each node. Every object view is drawn in 2D, meaning that we can assign two axis labels per object view and use transitions in the graph to track rotations. Figure 3.22 shows an abstract ED with three views and the axis labels assigned to them. Next, we select the top-left node as the source for a breadth-first search to obtain a list of transitions and assign axis labels to each node, starting with X and Y for the source node. Specific label names are not important, as domain experts will later have to decide which orientation best fits the chosen AM process. The largest lengths for the three axes can now be derived from the object view

bounding boxes. After that, we identify the longest dimension lines that best fit the respective extents for all three axes. These lengths are measured in pixels and therefore are not true to scale. Assuming a drawing has a $1:1$ scale, we can convert the pixel lengths to millimeters if the scan resolution (dots per inch) is known and would not have to rely on the OCR results. If we knew the real scale of a drawing, we could omit OCR even for arbitrary scales. However, we noticed that scale specifications in EDs are prone to variance in their appearance, size, and location. Moreover, detail views frequently include extra scale indications that differ from the overall scale of the drawing, which is why we decided not to attempt to identify scale indications directly, as this would go beyond the scope of this work.

EDs are drawn to a specific scale, thus, all dimension lines in a valid view grid $\mathbb{G}$ satisfy the following criterion:

$$\frac{l_{ocr}}{l_{px}} = c, \forall l \in \mathbb{G}, \tag{3.13}$$

where l_{ocr} is the numerical value of the assigned dimension number, l_{px} is the line length in pixels, and c is a constant. In other words, the ratio between the value predicted by the OCR model and the length of the dimension line in pixels must be the same constant for all lines in a grid. This fact allows us to run a simple optimization scheme to find the best fitting OCR-to-pixel ratio c, which should be more robust against faulty predictions than simply taking the raw OCR values. Additionally, it enables us to select the most likely OCR prediction out of the four options that result from processing each dimension number in four different 90° orientations. To achieve this, we pick the most likely longest lines for each main axis. Next, we generate a set of ratios by calculating the four ratios per line. We iterate over all lines and their four ratios and choose the current line-ratio combination as the reference value. We then calculate the squared error by comparing the reference ratio with all the other lines. This results in four errors per compared line, from which we select the minimum value. Summing up these errors of all compared lines gives us the score for the current line-OCR combination. We select the OCR-to-pixel ratio that achieved the overall lowest score. It is important to note that the characters in the OCR predictions may not be convertible to numerical values. In this case, we set them to a high value like 10000 when running the optimization so that these combinations produce large errors. Lastly, we multiply the longest dimension line lengths and longest object view bounding box lengths by the OCR-to-pixel ratio to produce the final external dimension output.

Special Cases A few special cases must be considered when extracting the external dimensions from EDs. The procedure works for drawings with two or more views, however, many drawings contain just a single view, in particular if they depict flat or rotational parts. The third dimension is implicit in many rotational cases, for example, given by a dimension number with a diameter symbol like Ø25. The thickness of flat parts is often denoted in a specific way such as $t = 5$, *thickness* 5, or *Dicke* 5. Therefore, we check all detected dimension numbers for the existence of these symbols or strings and assign the third dimension accordingly. If there are multiple hits, we choose the one with the largest

numeric value. Choosing the largest value is sufficient because domain experts will use the extracted external dimensions to assess whether an object fits in the building chamber of an AM device. Our ED pipe finds dimension lines with two arrowheads as starting and end points, but some rotational drawings show dimensions with only one arrowhead pointing towards the surface of the object. In this scenario, we examine all detected dimension numbers and check which ones could be linked to detected arrowheads in a similar way as we connected two arrowheads to obtain a first guess of dimension lines. Again, we select the one with the largest numeric value. Additional special cases are symmetric parts and split views of parts, which we consider to be beyond the scope of this thesis. The last special case is the depiction of bending parts, which often consists of a regular view grid that shows the object in its final shape and an unwinded flat representation. In most situations, the unwinded view drops out of the valid view grid, so it does not require additional processing.

Pipeline Output The final output of the ED pipe includes the external dimensions in pixels and millimeters, a set of detected surface finish symbols and welding symbols, a crop of the textfield, segmentation masks for the individual object views, and optionally, a set of all detected dimension lines. We will present more complexity measures that could be derived from our pipeline in Chapter 6 and Chapter 7.

CHAPTER 4

Implementation

In this chapter, we present details on the prototypical implementation of the ED pipe. We will cover the libraries used for various tasks in our prototype, such as data handling, image processing, model design, training and evaluation, graph processing, and output generation. Next, we give some details about the implementation. In addition, we describe the hardware utilized to train the segmentation models and briefly examine the efficiency of the pipeline in terms of processing time. The chapter concludes with examples of the format of pipeline outputs and information about the usage of the prototype. The prototype is currently used by our partners in the AM4Rail research project, ÖBB, Wiener Linien, Fraunhofer Austria, and Fraunhofer IAPT, to process a large number of real-world EDs with the goal of assessing the AM potential of the objects depicted in these drawings. They will use the various pipeline outputs to evaluate the possibility of additively manufacturing real components used in the railway sector with regard to technical and economical feasibility.

4.1 Libraries and Frameworks

In this section, we give an overview of the used technology stack. The implementation is written entirely in Python [pyta] and we employ widely used third-party packages for various tasks. Python is a high-level, interpreted programming language and is particularly popular in scientific computing, data analysis, and artificial intelligence. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, and features a comprehensive standard library. Furthermore, it potentially runs on many platforms and a large number of frameworks integrate well with the language. The use of notebooks enables us to interactively set up and run experiments and examine the results. Many standard machine learning frameworks support Python development, making it one of the most popular environments among AI researchers.

We use PyTorch 1.13.1 [pytb] with CUDA 11.7 [cud] as the basis frameworks for implementing, training, and running the segmentation networks. PyTorch is a flexible platform for deep learning research and development. It is known for its dynamic computation graph, which is particularly beneficial in research settings. Utilizing GPU acceleration through integration with CUDA, a parallel computing platform developed by NVIDIA, PyTorch improves computational efficiency, which is essential for training complex neural networks and processing extensive datasets. This integration facilitates rapid prototyping and experimentation, while the parallel processing capabilities of CUDA significantly reduce computation times by using many GPU cores simultaneously compared to CPU execution.

PyTorch Lightning is a lightweight PyTorch wrapper that aims to reduce the amount of boilerplate code when experimenting with AI algorithms while maintaining full flexibility [lig]. It provides us with a structured environment for model training, evaluation, and inference, making code more readable, reusable, and scalable. PyTorch Lightning integrates seamlessly with TensorBoard [ten], a visualization toolkit for machine learning experimentation that offers a rich set of tools to track metrics and losses, as well as inspect model graphs. This is why we choose PyTorch Lightning to encapsulate the models and to track our experiments. Moreover, PyTorch Lightning enables us to implement scalable data loaders and processors that are not only utilized for training but can be easily adjusted and reused for inference purposes.

Segmentation Models PyTorch [seg] is a library for DL segmentation models, offering a unified interface for working with various architectures, such as U-Net, FPN, and DeepLabV3. It simplifies the process of implementing and training segmentation models by providing default modules with pre-trained weights. This allows for experimentation with different architectures, significantly accelerating the development cycle for projects involving image segmentation tasks. The library is designed to be compatible with PyTorch and leverages its features to ensure efficient training and inference. We use the library to implement the models for all segmentation tasks in the ED pipe.

NumPy [num] is a fundamental package for scientific computing in Python, offering support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these structures. PyTorch is compatible with NumPy through a dedicated bridge that allows torch tensors on the CPU and NumPy arrays to have common memory locations, ensuring that modifications in one will reflect in the other [pytc]. SciPy [scib] is built on NumPy by adding a collection of algorithms and high-level commands for data manipulation, signal processing, optimization, and statistical analysis. We use Scikit-Learn [skl], which is built on SciPy and NumPy, to perform PCA on segmented arrowheads, among other things.

Shapely [sha] is a Python package for set-theoretic analysis and manipulation of planar geometric objects, based on the widely adopted GEOS (Geometry Engine - Open Source) library. It provides us with tools to create, manipulate, and analyze complex geometric shapes, such as lines, points, and polygons, facilitating operations such as merging, intersecting, and finding differences between shapes. The library simplifies tasks that

involve handling spatial relationships, such as proximity, containment, and overlay operations. We make extensive use of Shapely and its capabilities in the dimension line processing stages of the pipeline, for example, when checking intersections of entities.

OpenCV [ope] is a comprehensive library of algorithms mainly aimed at computer vision and image processing. It is one of the most widely used computer vision libraries with support for many programming languages, including Python. The Python version integrates well with NumPy data structures, which is why we use it extensively for image processing tasks such as color conversions, rescaling, connected component labeling, morphological operations, and thresholding, among others. In addition, we use Scikit-Image [scia], which builds on the foundations of SciPy and NumPy, to detect contours in binary masks.

Albumentations [alb] is a fast and flexible image augmentation library for Python, designed to augment and transform images for machine learning and deep learning tasks. It supports a wide range of augmentation techniques, such as flipping, affine transformations, noise injection, and color adjustments, which can significantly enhance the diversity of training datasets. Albumentations supports data structures from popular DL frameworks such as PyTorch and TensorFlow. We use the library in all training configurations for the segmentation models to artificially diversify the limited training data set.

We use the Hampel [ham] Python package to use a Hampel filter for removing outliers in the dimension line processing stage of the ED pipe. It is a lightweight package with a robust implementation and good documentation.

MMOCR [KSL+21] is an open source toolbox based on PyTorch for text detection and recognition in images. It supports a wide range of state-of-the-art algorithms for both printed and handwritten text, making it suitable for a variety of OCR applications. The toolbox is designed to be modular and extendable, allowing researchers to experiment with different components and customize their OCR systems. We use MMOCR to fine-tune an OCR model and to perform OCR on image crops containing dimension numbers.

NetworkX [net] is a Python package for the creation and manipulation of graphs. It provides tools for working with graphs and networks, offering data structures for graphs, digraphs, and multigraphs, along with a large number of functions that operate on these structures. The library supports various standard graph algorithms, such as shortest path, clustering, and circle detection. In addition, it facilitates the visualization of graphs. We use NetworkX in the dimension line processing stage to untangle clusters of lines and to remove false positives.

PDF2Image [pdf] is a Python module that converts PDF files into images. We use it to render portable document format (PDF) files for further processing at the beginning of our pipeline. Tifffile [tif] is a Python library that provides a way to read and write TIFF files, including support for large multi-dimensional arrays of image data stored in TIFF format. Many ED scans of our partners are stored in TIFF format, so we need a tool to

open and interpret them reliably. OpenCV can also open TIFF files, but often interprets them incorrectly, leading to strange artifacts or exceptions.

We hand over a functional prototype to our research partners and aim to reduce the amount of work they need to put in to set up the system on their devices, which is why we choose to bundle the entire application into a single executable file. PyInstaller [pyi] is a tool that converts Python applications into standalone executables, under Windows, Linux, and Mac OS X. It is particularly useful for distributing Python programs to end-users who may not have Python installed on their systems, as it packages the Python interpreter and all the dependencies into a single executable file. This simplification significantly improves the accessibility and portability of the applications. Auto-Py-to-Exe [aut] is a graphical interface for PyInstaller, making the process of converting Python scripts into executables more user-friendly. It provides a straightforward interface that allows us to select the desired options and then compile the application.

4.2 Implementation Details

The prototype is implemented in Python using the packages mentioned in Section 4.1. Since it is only a prototypical implementation, we do not focus on execution speed or optimal memory efficiency, but rather on creating a functional application that our research partners can use. We are aware of some potential improvements and address them in the discussion chapter. We base the implementation on types that only encapsulate data without methods and use Python's dataclass annotation for this purpose. We implement dataclasses for the following entities:

- Params
- DimensionNumber
- View
- ObjectSolid
- Arrowhead
- DimensionLine

The Params type stores the overall configuration of the pipeline, including all hyperparameters such as tolerances and thresholds, output configuration, and the paths to the stored weights of the four segmentation models. The other types contain all the necessary information for an instance of a particular entity obtained by processing the respective segmentation output or gathered during further processing stages in the pipeline. All entities have a unique index and store additional geometric properties, including the center point, the area in pixels, the bounding box coordinates, and a polygonal representation of the instance in the form of a shapely.polygon object. The polygon representation allows

Model	Training Duration
Textfield	4 hours
View	21 hours
Object View Fine	21 hours
Arrowheads, Numbers, Symbols	80 hours

Table 4.1: Training times of the individual segmentation models on consumer hardware.

us to utilize the functions provided by Shapely to efficiently perform intersections and other operations. These operations are a key building block of the ED pipe and are used to link associated entities and remove outliers. Arrowheads and DimensionLines have additional members. An arrowhead object stores its orientation and lists of associated views, object views (ObjectSolid), and dimension numbers. A dimension line object references two arrowheads and stores the line length, direction, and corrected endpoints. In addition, it contains a list of potential linked dimension numbers and a list of other dimension lines that it covers.

We use the PyTorch-Lightning infrastructure to implement and train the segmentation models. Every model extends a pl.LightningModule class and we make use of the save_hyperparameters method to track hyperparameters of experiments. The pl.LightningModule allows us to encapsulate not only the model itself, but also to define methods for different tasks, such as training, validation, or inference. Furthermore, it can encapsulate the setup of the training optimizer and LR schedulers. The data loaders are also implemented to fit into the PyTorch-Lightning ecosystem by extending the pl.Dataset and pl.DataModule classes, again tracking all hyperparameters. This removes boilerplate code and allows us to visualize training runs in TensorBoard. The models are trained by utilizing a pl.Trainer and providing it with the pl.LightningModule and pl.DataModule. Moreover, loading a trained model for inference and setting up the correct data loader and hyperparameters is easier and more reliable when using the functionality provided by PyTorch-Lightning.

4.3 Hardware and Performance

The segmentation models were trained on a consumer PC with an AMD Ryzen 7 5800X 8 core processor, 32 GB of RAM, and an NVIDIA RTX 3080 Ti GPU with 12 GB of VRAM and 10240 CUDA cores. The limiting factors during training were the 12 GB of VRAM that limited batch size when optimizing the MiT-FPN architecture, and the data loading, processing, and augmentation speed, due to the 8 core CPU. A workaround for the small batch size is the accumulation of gradients over multiple batches before performing an optimization step, which is what we used when training all segmentation models to achieve a virtual batch size of 32 or 64. Table 4.1 shows the training times of the selected segmentation models in hours. It is noticeable that the textfield segmentation model needs significantly less time for training than the others. This is because it only trains on resized versions of the full input sample and we can fit the entire training set

into memory which reduces loading times. Arrowhead, dimension number, and symbol segmentation takes by far the longest time to train because of the very large size of the training data set, which is much larger than, for example, the view training data set. This also makes it impossible to load all samples into memory for fast loading and processing, since we are limited to 32 GB RAM.

The time required to run the entire pipeline depends on the complexity and size of the input ED. Some EDs contain many arrowheads, sometimes exceeding 150, which causes the processing to take longer due to performance bottlenecks in the dimension line processing stages of the pipeline. It would be possible to parallelize some of the tasks for a significant improvement in speed, however, this would be beyond the scope of this thesis. Another performance bottleneck is the inference of the arrowhead, dimension number, and symbol segmentation model. As described in Section 3.3, we perform inference on overlapping patches of the input drawing at multiple scales. The MiT backbone in our model scales quadratically to the input size, significantly slowing down the inference of larger patches. If the input drawing has high resolution, it is split into a larger number of patches. Furthermore, combining the predicted patches into the final output prediction also takes longer. These problems could be alleviated by parallelization, which is again not in the scope of the implementation. Running the pipeline for an ED of average complexity and a resolution of roughly 50 megapixels takes around 10 to 15 seconds in total. We tested a worst-case scenario with a 200 megapixel ED containing over 250 arrowheads and it took around 75 seconds to run the pipeline. We will discuss possible improvements in the discussion chapter.

4.4 Pipeline Outputs

The main output of the ED pipe is a JSON file containing the external dimensions of the object depicted in the input ED. In addition, we output the results of the textfield segmentation to be used for further processing by our research partners. Optionally, we output all detected dimension lines in another JSON file as well as the results of the object view segmentation.

The file containing the external dimensions has the following structure:

```
{
    "line_x": longest line in X (mm),
    "line_y": longest line in Y (mm),
    "line_z": longest line in Z (mm),
    "object_x": longest object bounding box edge in X (mm),
    "object_y": longest object bounding box edge in Y (mm),
    "object_z": longest object bounding box edge in Z (mm),
    "ratio": conversion ratio from pixels to millimeters,
    "max_val": maximum length found with OCR (mm),
    "assigned_axes": list with axes that were assigned,
    "diameters": list of detected diameters (mm),
    "thickness": list of detected thicknesses (mm),
    "all_x_lengths": list of all lengths in X (pixels),
    "all_y_lengths": list of all lengths in Y (pixels),
    "all_z_lengths": list of all lengths in Z (pixels),
    "max_x_length": overall longest line in X (mm),
    "max_y_length": overall longest line in Y (mm),
    "max_z_length": overall longest line in Z (mm)
}
```

The textfield segmentation output consists of a number of crops from the input drawing where textfields were found, as can be seen in Figure 4.1. In addition, we output the bounding box coordinates of each detected textfield in a JSON file. The object view outputs and the outputs of the surface finish and welding symbols are structured in a similar way.

4.5 Deliverable and Usage

The deliverable is a simple command-line application packaged with PyInstaller and auto-py-to-exe. It does not require the installation of additional software and can be run directly by our research partners. We include functionality to run the pipeline on a single input ED and on a folder containing EDs. If the application runs on an entire folder of drawings, it creates an output folder for each input to store the respective outputs. Finally, we include the ability to run textfield segmentation only, as it is much faster than running the entire pipeline. This works on single drawings as well as on input folders in a similar way. The command-line application requires the path to the input file or the input folder and a path where the outputs should be saved. In addition, optional arguments can be set to output textfields only, object views, and all detected dimension lines. The application was tested by our research partners and they reported that it was working without issues. If an error occurs during execution, a log file is created with

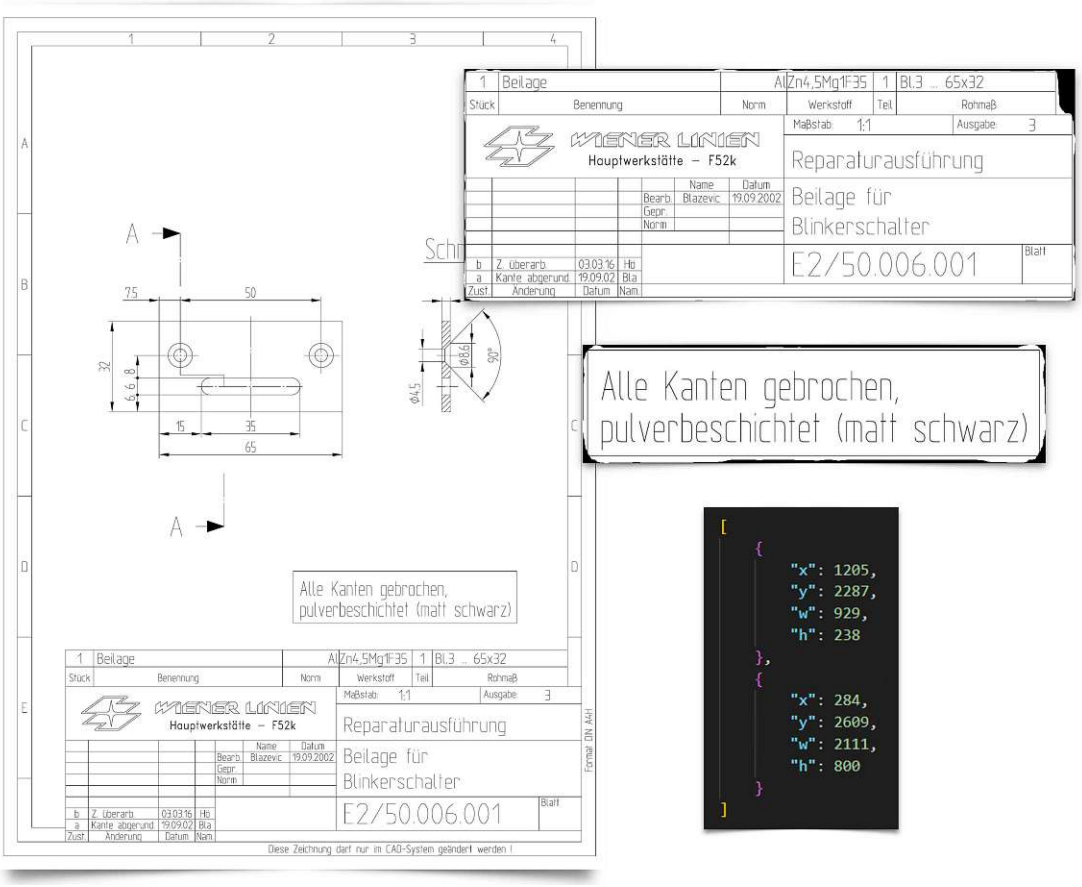


Figure 4.1: The textfield segmentation algorithm outputs crops and bounding boxes of all detected textfields in an ED for further analysis by our research partners. ED source Wiener Linien [wl].

details about the problem, such as the stack trace. We will receive the logs at some point in the future and analyze them.

CHAPTER 5

Results

In this chapter, we present the results of the prototypical implementation of our ED pipe. We present qualitative examples of all pipeline stages and conduct a basic quantitative evaluation. In general, the pipeline produces satisfactory results on the test data set, indicating that it generalizes well to different styles of EDs found in real-world databases. We evaluate the average DSC based on the formula in Equation 3.3 for every segmentation model, to obtain an estimate of the performance of each model on the test data set. Furthermore, we evaluate the end-to-end performance of the ED pipe in terms of percentage differences from predicted external dimensions to true external dimensions.

5.1 Quantitative Evaluation

Segmentation Models Evaluation The test sets for textfield, view, and object view segmentation contain all 90 examples from the global test set. The arrowhead, number, and symbol segmentation test set contains only a subset of 30 examples from the global test set due to time-consuming annotation. Table 5.1 shows the evaluation metrics of the segmentation models. The segmentation models seem to perform well on the test set based on the quantitative evaluation results, with all scores relatively close to the

Model	**Mean DSC↑**
Textfield	0.98079
View	0.93887
Object View	0.95044
Arrowheads, Numbers, Symbols	0.92943

Table 5.1: Quantitative evaluation of the segmentation models used in our ED pipe. The models show good overall performance based on the measured mean DSC scores. The maximum value of the DSC metric is 1.0, larger value represents a better performance.

DSC score maximum value of 1.0. The textfield model achieved the best score, which makes sense because textfields are typically separated from other entities in an ED, have a distinctive tabular structure, and are often positioned at the borders of a drawing. Moreover, the textfield model receives the complete drawing as the input context to make predictions. The view model achieves a lower score due to the less distinct borders of the views, often leading to ambiguity in determining where one view ends and another one begins. Due to this uncertainty, a lower score is expected, as opposed to the textfield model. The object view model scores slightly higher than the view model, which we attribute to the better defined boundaries and the additional simplified input where a lot of clutter was removed. In addition, we perform inference with the object view model at multiple input scales, which also improves its overall performance. The arrowhead, dimension number, and surface finish symbol segmentation model reached the lowest score of all the models, but still performs well on the test data set. This segmentation task is arguably the hardest one to train due to the large variety and relatively small size of the segmentation targets. The lower score may also be caused by the style of our annotation. Dimension numbers have loose borders in the annotation similar to views. Furthermore, a pixel-perfect annotation of arrowheads and surface finish symbols was not possible due to time constraints and the sheer number of arrowheads in the samples, so they are only annotated roughly, which may introduce a larger error.

External Dimensions Evaluation We evaluate the performance of the end-to-end pipeline by calculating the mean of the absolute percentage prediction error (PPE) [GBF95] between the predicted external dimensions and the true external dimensions. It can be formulated as follows:

$$\frac{1}{N}\sum_{n=1}^{N}\left|\frac{\hat{y}_n - y_n}{y_n}\right|, \tag{5.1}$$

where N is the number of external dimensions in X, Y, and Z for all samples in the test set, $\hat{y}$ is the predicted value, and y is the true length of the respective dimension. This results in the mean percentage deviation of the pipeline results in comparison to the ground truth, which is a more meaningful value in this case than, for example, a mean squared error. Each sample in the test set is annotated with its three external dimensions in X, Y, and Z, and then compared with the corresponding maximum value along each axis in the pipeline prediction, taking into account diameters and thicknesses if the third axis is not assigned in the prediction. We remove all assembly drawings and not fully dimensioned drawings from the test set, leaving us with 84 examples. The OCR stage of the pipeline failed to deliver correct results in three of the 84 samples (3.57%). Since this work does not aim to maximize OCR performance and the OCR stage could easily be replaced with a better suited model, we exclude the failed cases from the final score calculation. The ED pipe achieves a mean absolute PPE score of **0.731%** on the remaining 81 samples, meaning that the deviation from the ground truth is on average less than one percent in all axes. This score could be further improved by searching for the best matching OCR result for each external dimension instead of multiplying the pixel length with the OCR-to-pixel ratio. However, this approach would fail more

often if an unsuitable OCR model was used in the pipeline. We did not implement and evaluate this feature due to time constraints. Moreover, utilizing the OCR-to-pixel ratio has the advantage that the OCR model does not have to be perfect in order to generate satisfactory predictions of the external dimension.

5.2 Qualitative Evaluation

In this section, we present qualitative results for each processing stage when running the entire pipeline end-to-end on three exemplary EDs. In addition, we present some selected examples of other drawings to illustrate certain advantages and limitations of our approach.

Example 1 Figures 5.1, 5.2, 5.3, 5.4, and 5.5 depict the results of each pipeline stage for the first example drawing. It is an ED generated with a CAD tool in 2002 with good image quality. The image was exported directly from the CAD software, hence it does not suffer from scan artifacts. All important entities are clearly visible, but the various line types are difficult to distinguish. The dimension and extension lines are thicker than the visible object outlines, which is the opposite of what is expected. On the other hand, the arrowheads have an ideal shape, the text of the dimension numbers is legible, the depicted object is dimensioned fully, the views are clearly separated, and the title block is positioned in the expected location at the bottom right of the frame. The original drawing is shown in Figure 5.1 (a). The early stages of the ED pipe perform image segmentation tasks, the results are given in Figure 5.1 (a,b,c). The textfield segmentation performs well on this input, finding areas with clear boundaries around the relevant parts of the drawing. The view segmentation also delivers good results but detects one false positive region that denotes the section through the object. This may be due to the text being emphasized with a line that looks similar to object outlines. The object view segmentation correctly detects both relevant views with acceptable detail, with the thinner object view showing a bit of over-segmentation, which is probably caused by the structure of the angle measurement. For our purposes, these results are sufficient as a starting point for extracting the external dimensions of the input ED. Figure 5.2 shows the results of the arrowhead and dimension number segmentation, with a crop of the respective area in the original drawing in (a) for readability. The arrowhead segmentation result is of high quality and every target was detected accurately. All dimension numbers were also detected, but there is a bit of ambiguity, particularly in the vertical chained dimension on the left side of the drawing. Here, the dimension numbers are close together, resulting in a single merged region in the segmentation. On the right side, we also see a small false positive region. Next, the pipeline uses the detected entities to estimate a raw set of dimension lines, which can be seen in Figure 5.3 (a). It is obvious that it contains some false positive dimension lines that cross each other and connect arrowheads that do not belong together. This happens because of the relatively high tolerances set as hyperparameters in this processing stage to account for different styles of EDs. The next stage in the pipeline untangles the estimated dimension lines and removes outliers. The resulting cleaned set of dimension lines is depicted in Figure 5.3 (b). It worked well in this

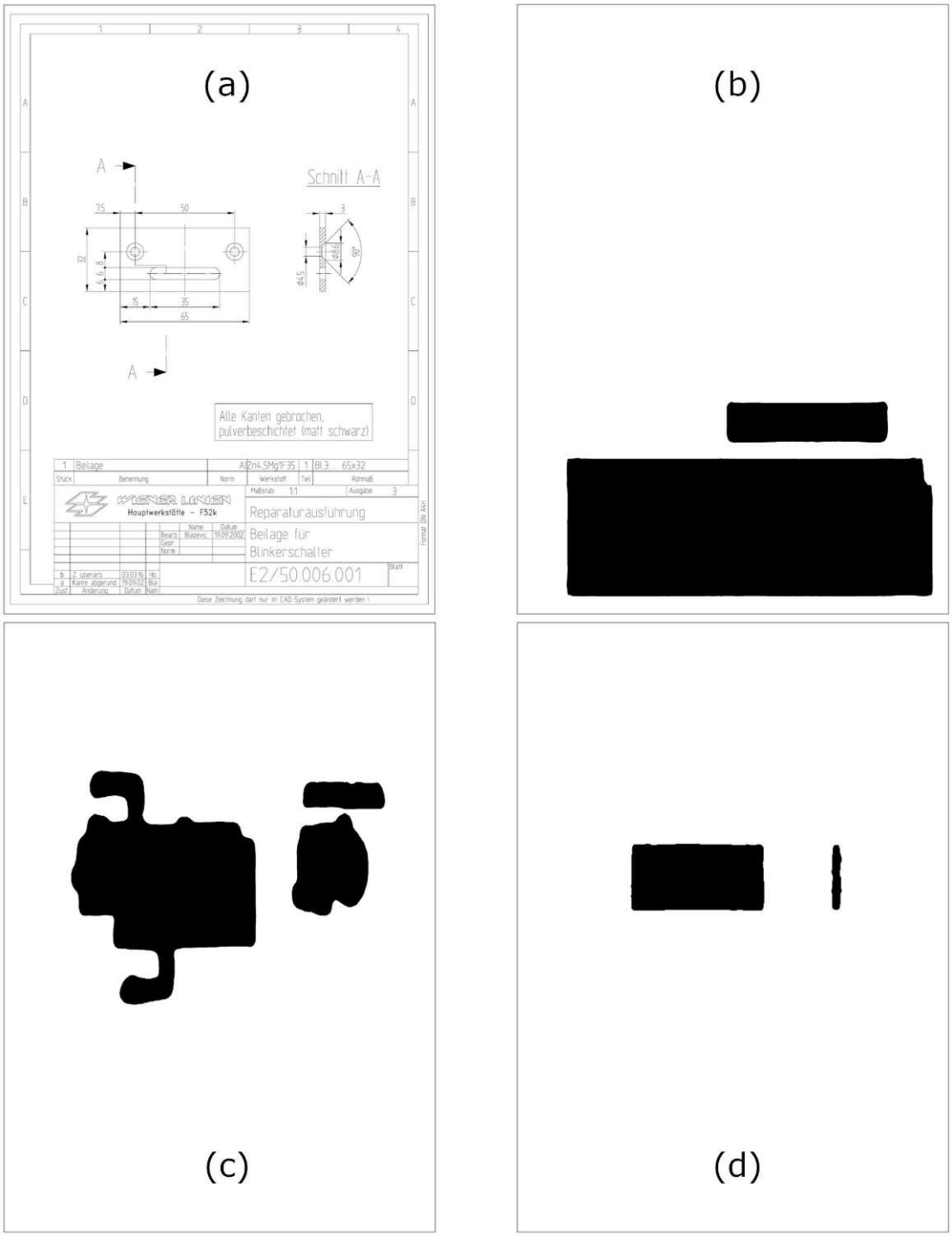


Figure 5.1: End-to-end processing of the first example drawing (figure 1 of 5). (a) Input image; (b) text-field segmentation result; (c) view segmentation result; (d) object view segmentation result. ED source Wiener Linien [wl].

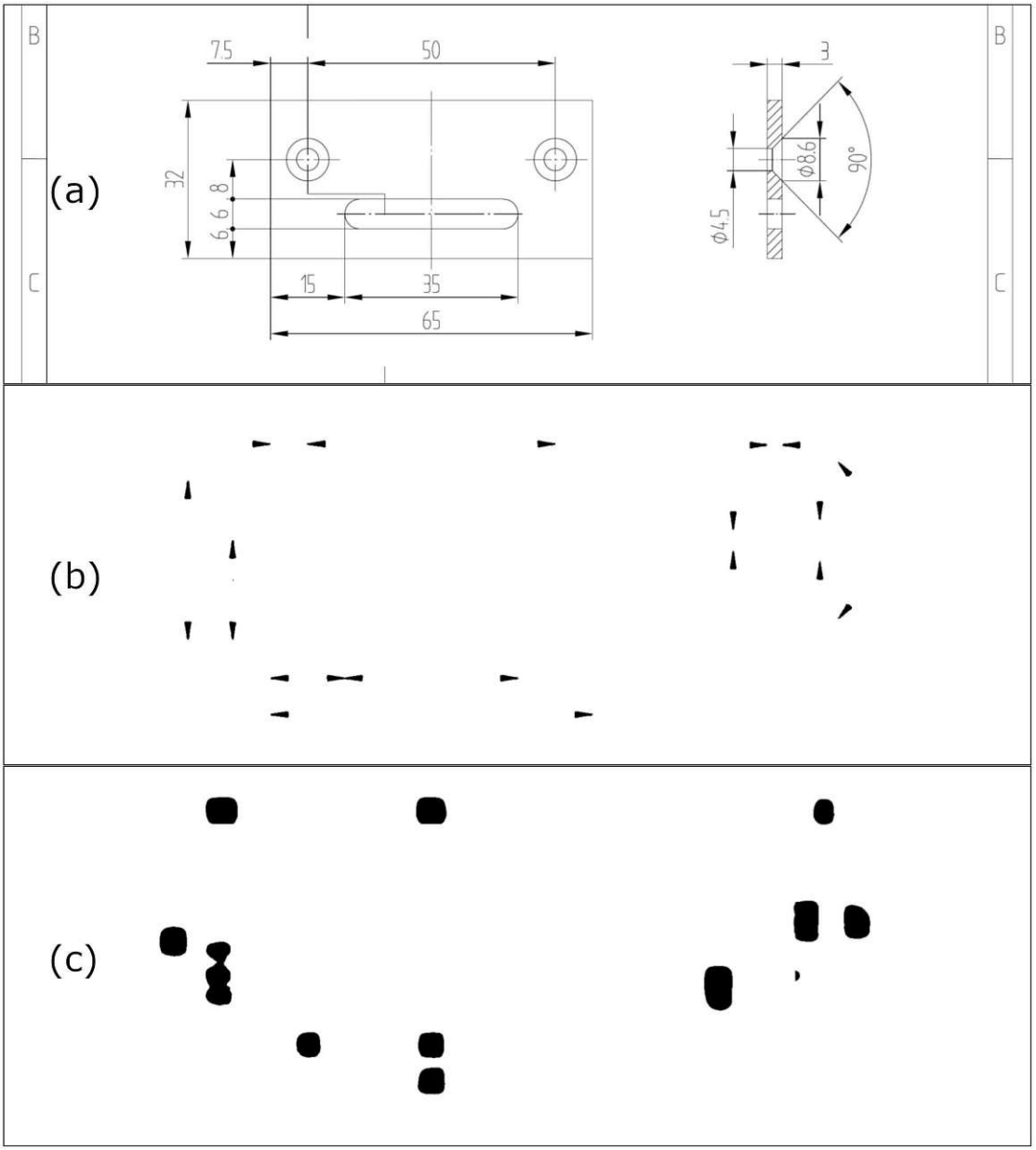


Figure 5.2: End-to-end processing of the first example drawing (figure 2 of 5). (a) Input image crop; (b) arrowhead segmentation result; (c) dimension number segmentation result. ED source Wiener Linien [wl].

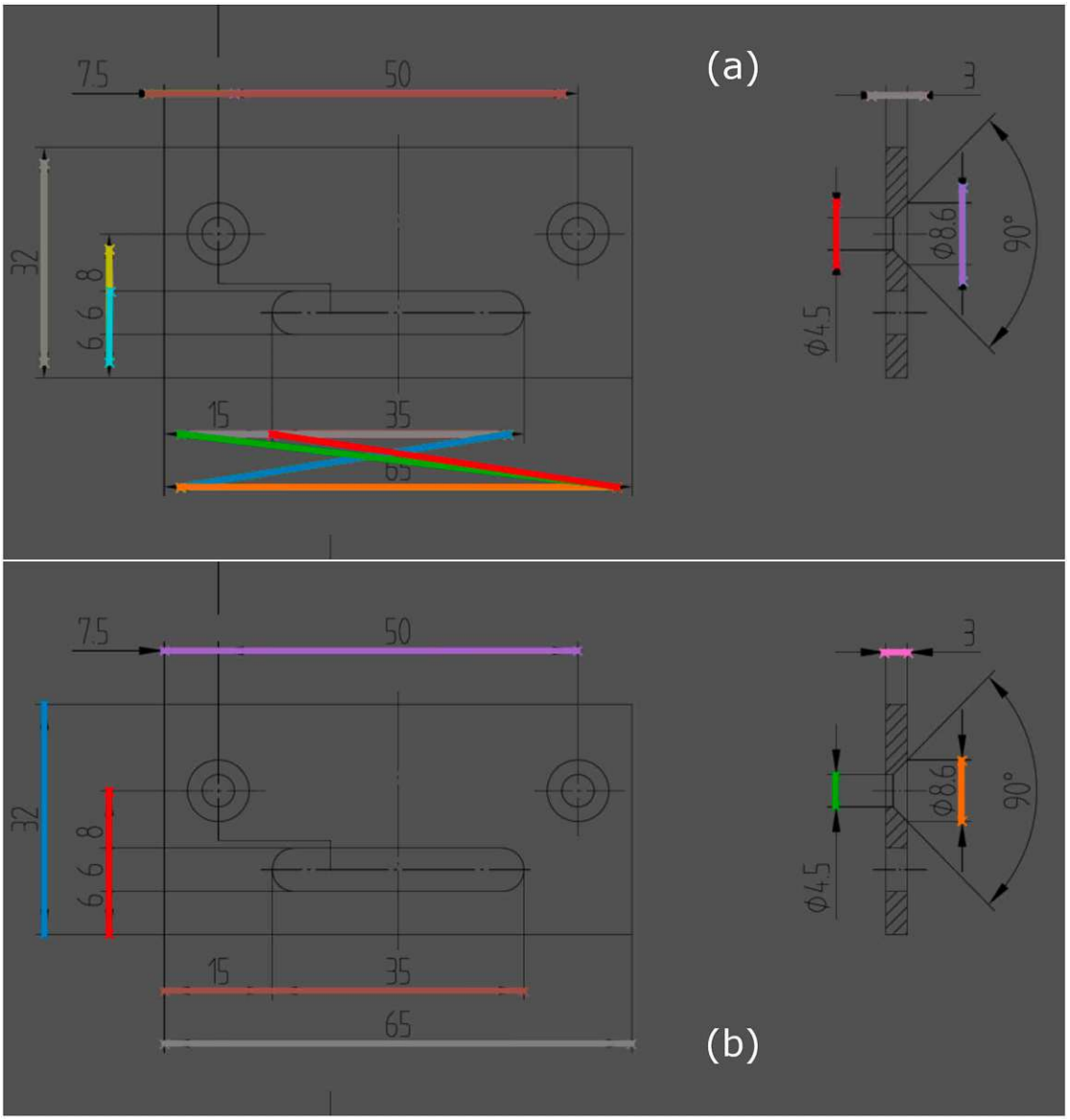


Figure 5.3: End-to-end processing of the first example drawing (figure 3 of 5). (a) Set of raw dimension lines before line processing; (b) filtered set of dimension lines after line processing. ED source Wiener Linien [wl].

example, removing all false positives and keeping the correct lines. Chained dimensions are substituted with a single line, as depicted in the image. Figure 5.4 displays the view grid map in (a) and the detected separate views with bounding boxes and adjacency information in (b). The algorithm found the correct view grid structure without problems. The final step in the ED pipe is the assignment of axis labels and the external dimension output, which is shown in Figure 5.5. The assignment for the first view can be seen

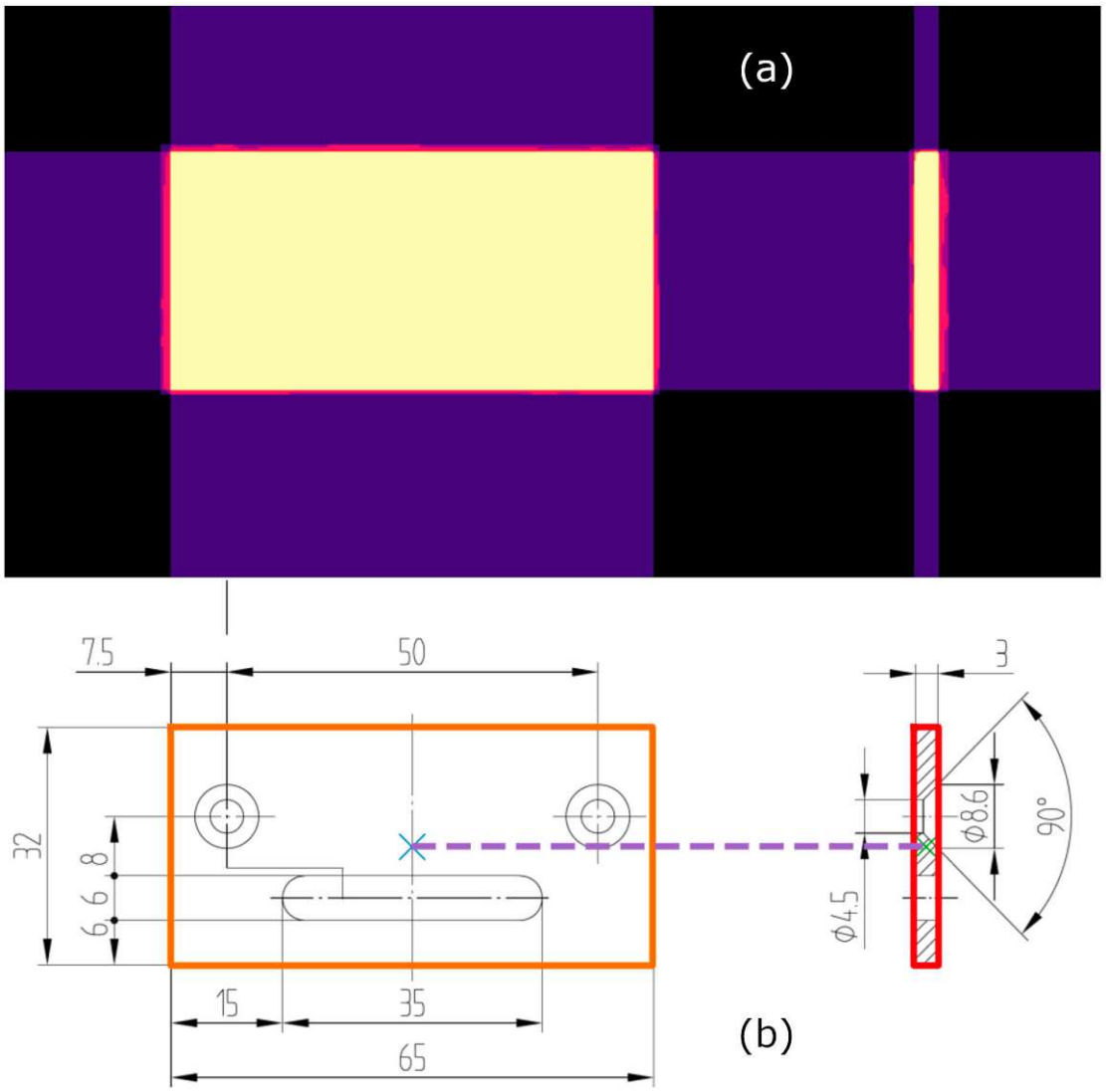
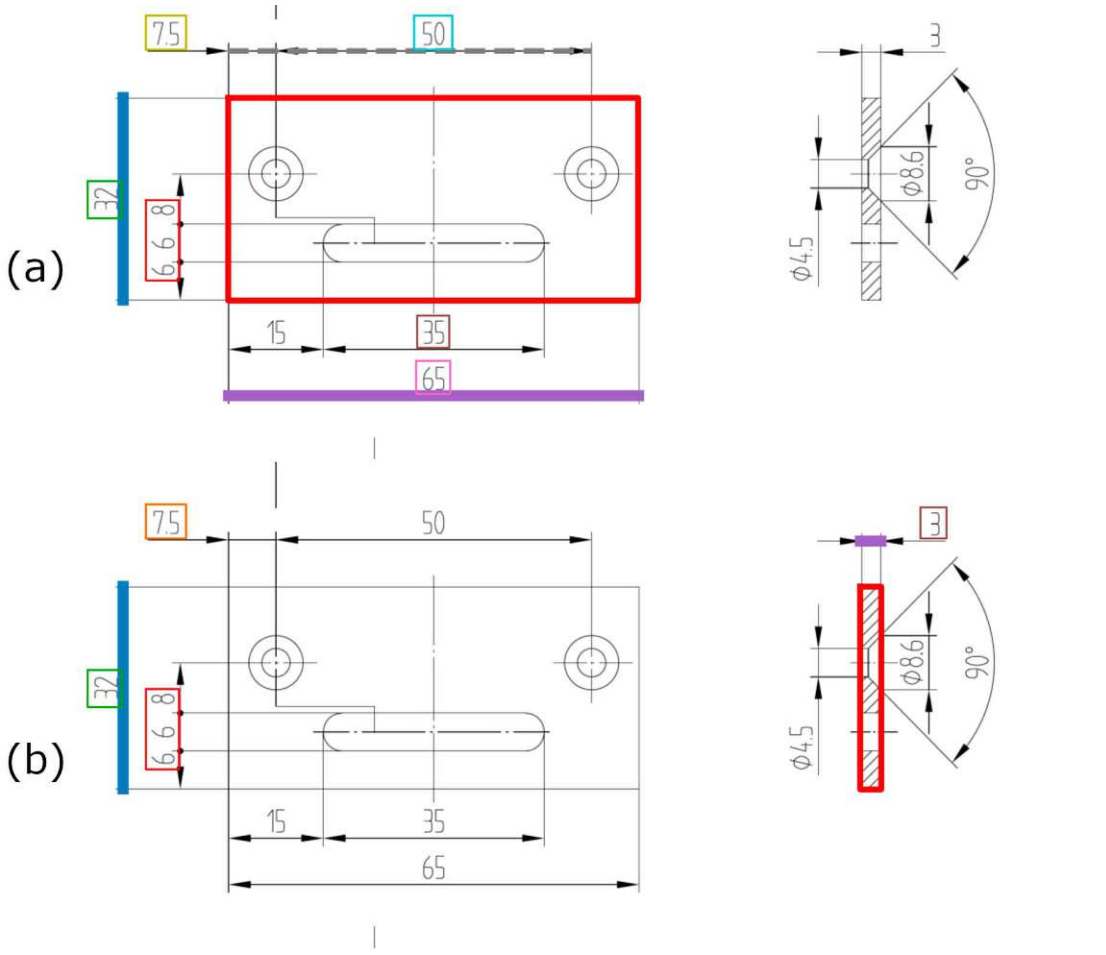


Figure 5.4: End-to-end processing of the first example drawing (figure 4 of 5). (a) View grid map; (b) view grid adjacency illustrated by the dashed line. ED source Wiener Linien [wl].

in (a) and for the second view in (b). The algorithm correctly chooses the external dimensions in X (65), Y (32), and Z (3). An excerpt of the external dimensions output after multiplying the line lengths with the OCR-to-pixel ratio is given in (c). We note that the resulting external dimensions are satisfactory and within the expected PPE tolerance based on the quantitative evaluation.

Example 2 In this example we process a hand-drawn ED from 1967 with a very different style than the drawing in the first example. The results of the individual pipeline stages are shown in Figures 5.6, 5.7, and 5.8. The image has severe scan artifacts and drawing inaccuracies, for example, missing and vanishing lines, as is visible in Figure 5.6. The title block covers a large area of the drawing and is located at the bottom of the frame. The textfield segmentation yields a satisfactory result, although, it has more difficulties than in the previous example, resulting in some holes in the title block. The holes occur in insignificant blank regions and are therefore not considered for further processing. The view segmentation delivers a good result, covering the expected image area. Interestingly, it attends to the stains visible in the top right of the drawing and includes them in the view prediction. Given the circumstances, the object view segmentation produces an acceptable result that is sufficient for further processing in the pipeline. Figure 5.7 depicts a crop of the original drawing in (a), the arrowhead segmentation result in (b), the dimension number segmentation result in (c), and the simplified version of the input drawing in (d). As in the first example, we see satisfactory results with some merging and over-segmentation occurring in the dimension number segmentation. The simplified version, which is used as a secondary input to the object view segmentation, is an uncluttered version of the original. The algorithm was able to successfully remove most of the structures that are not part of the object view. Figure 5.8 displays the estimated and refined dimension lines in (a, b), the view grid map and axis assignment in (c, d), and the excerpt of the external dimension output in (e). The single view was detected and the line processing removed all false positives. Since we only have a single object view, the third dimension needs to be determined. We detect diameter symbols and therefore know that the third axis must have the same length as the second one (30Ø). This example illustrates that the system is capable of handling such challenging cases where all dimension lines that typically connect arrowheads are missing.

Example 3 In the final end-to-end example, we process a hand-drawn ED from 1985. It is of better quality than the second example, but it also suffers from scan artifacts. Some lines are present, while others are missing or very faint and fragmented. Figures 5.9, 5.10, 5.11, and 5.12 show the results of the individual pipeline stages. All pipeline stages produce good results. It is important to note that the third dimension, although fully determined, is actually measured by a chained dimension as can be seen in the bottom right corner of Figure 5.12 (b). If we relied only on OCR, we would not be able to correctly determine the length of the third external dimension, which is obtained by adding the values included in the chained dimension 3 mm + 9 mm = 12 mm. However, our approach is capable of producing an approximate result of 11.865 mm which is close to the true length of 12 mm.



(c)

```
{
    "line_x": 65.0,
    "line_y": 32.09,
    "line_z": 3.17,
    ...
}
```

Figure 5.5: End-to-end processing of the first example drawing (figure 5 of 5). Assignment of axis labels and associated dimension lines to each detected view in the grid to output the external dimensions of the object depicted. The final values for the external dimensions are calculated by multiplying the line length in pixels by the OCR-to-pixel ratio (c). ED source Wiener Linien [wl].

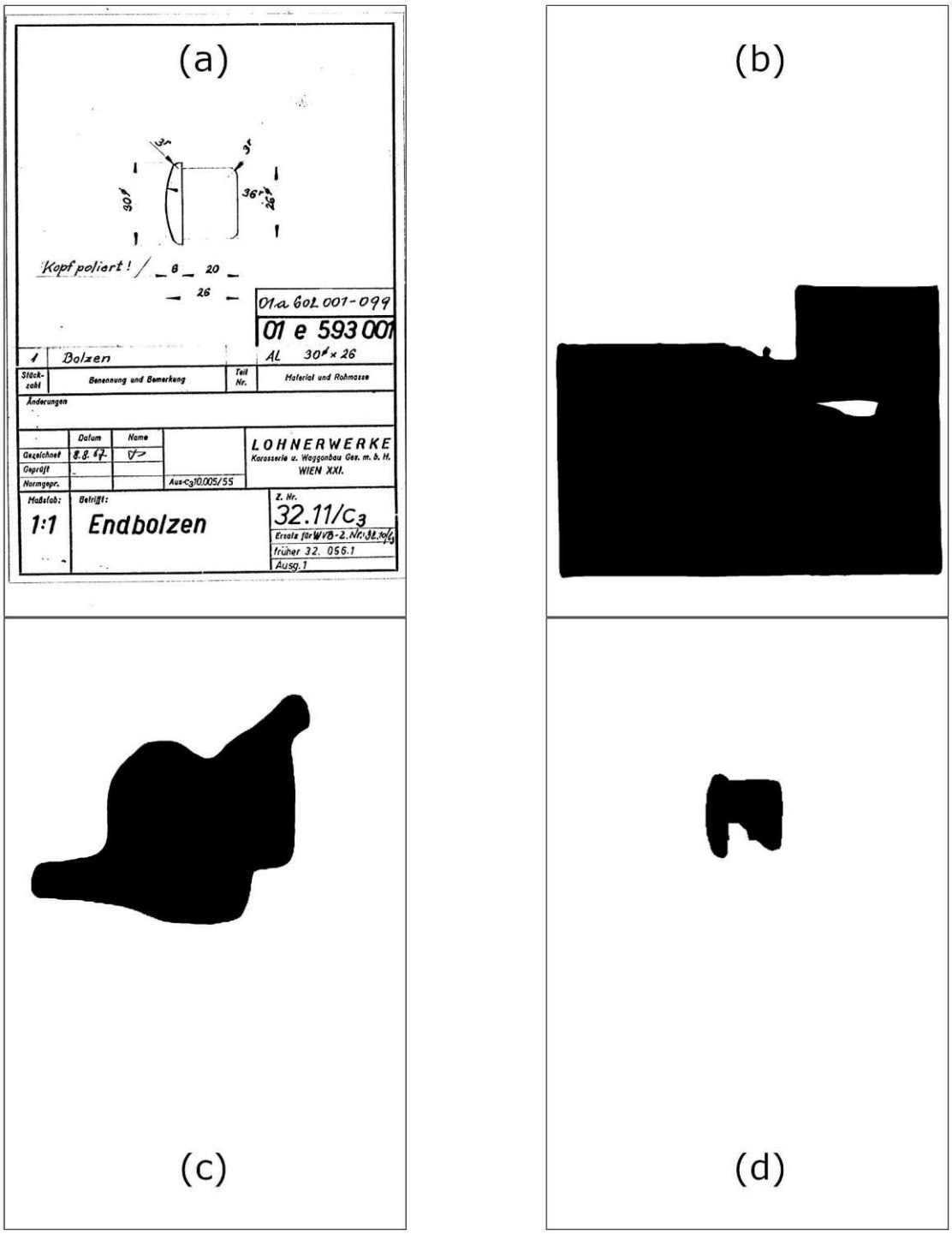


Figure 5.6: End-to-end processing of the second example drawing (figure 1 of 3). (a) Input image; (b) text-field segmentation result; (c) view segmentation result; (d) object view segmentation result. ED source Wiener Linien [wl].

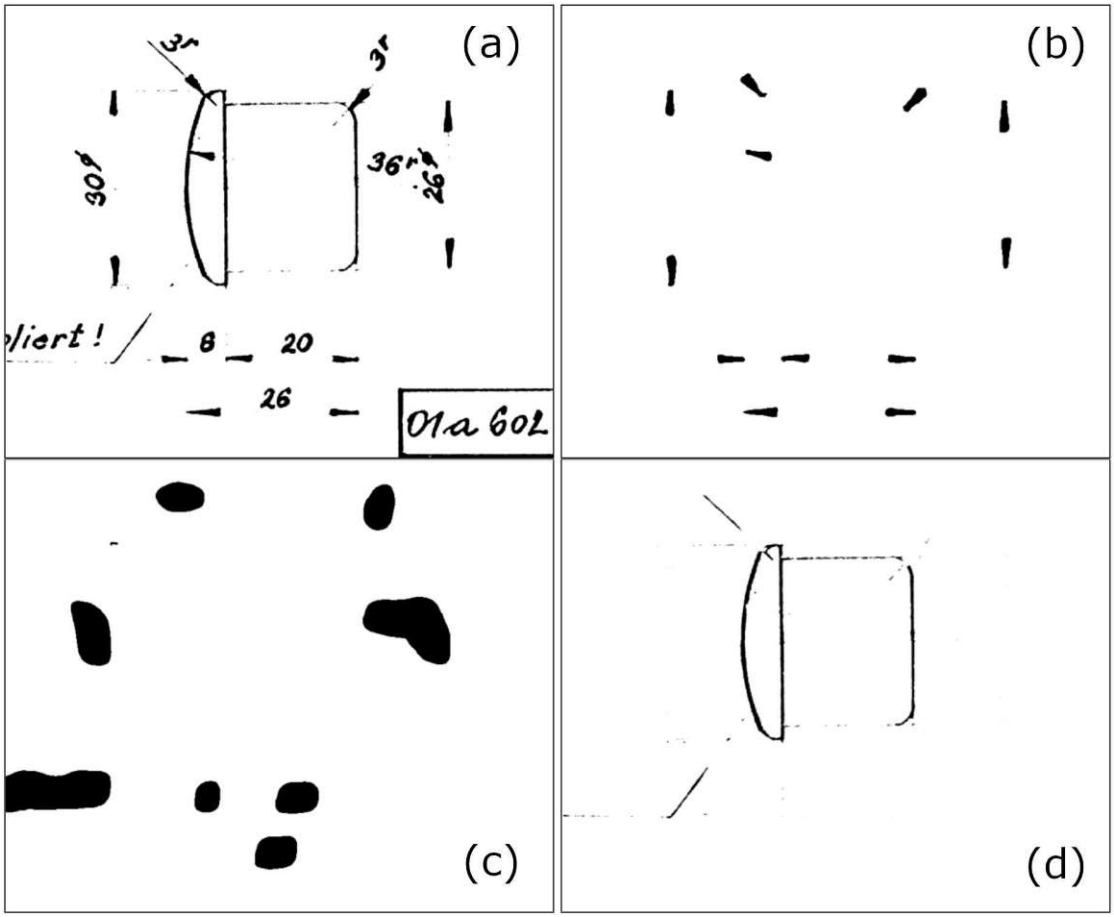


Figure 5.7: End-to-end processing of the second example drawing (figure 2 of 3). (a) Input image crop; (b) arrowhead segmentation result; (c) dimension number segmentation result; (d) simplified input image for object view segmentation. ED source Wiener Linien [wl].

Selected Examples We will now briefly present selected examples of other drawings without showing the output of every pipeline stage. Figure 5.13 gives an example of an ED in which the depicted object is not fully measured when looking at the dimension lines alone. This is reflected in the final output, which is listed in Figure 5.13 (c). Here, line_x, line_y, and line_z are all smaller than their counterparts, object_x, object_y, and object_z. The final dimension lines are depicted in Figure 5.13 (a) and the object view segmentation in Figure 5.13 (b). It is obvious that the object views cover a larger extent in all axes than the individual dimension lines. Figure 5.14 underlines the ability of the line processing stage to untangle clusters of lines and remove outliers even in cluttered and complex situations. Figure 5.15 depicts a more complicated view grid structure. It consists of three views that form the main grid, an unwinding at the bottom right, and an isometric view at the top right that does not belong to any grid. The

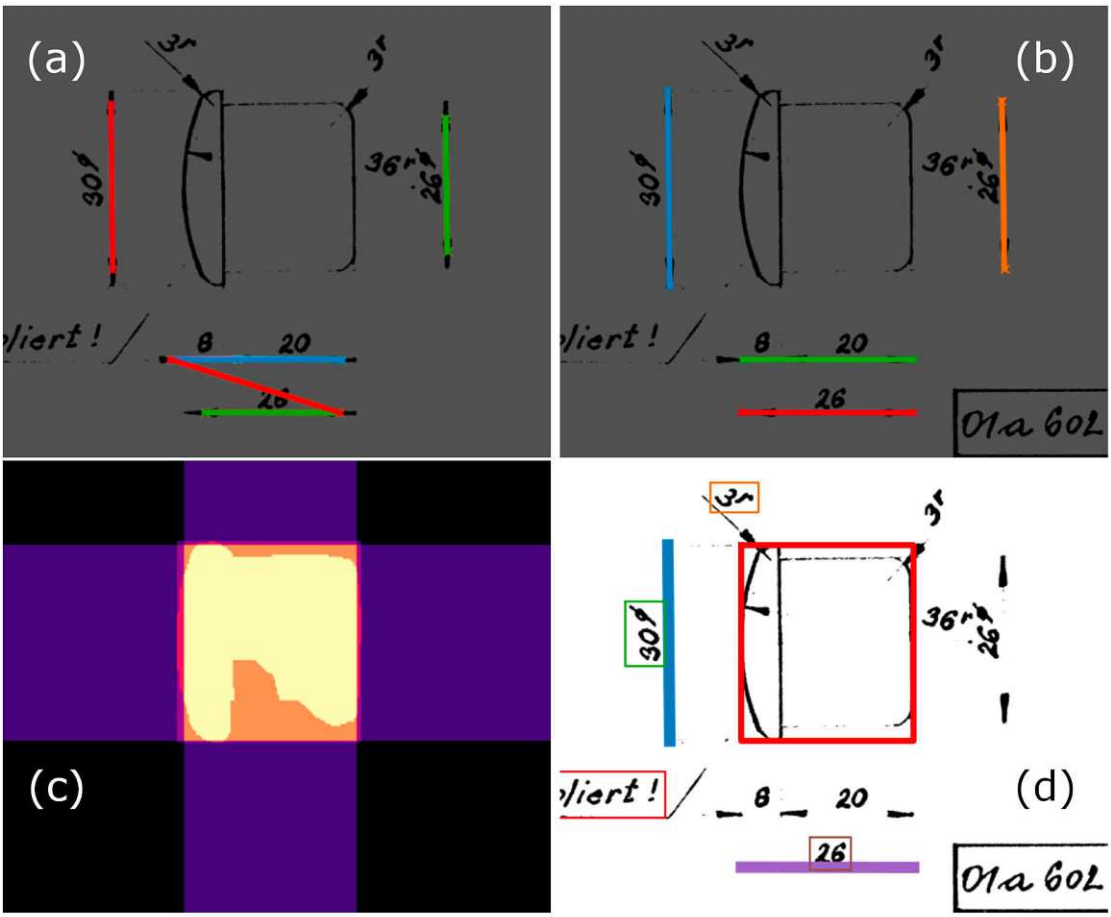


Figure 5.8: End-to-end processing of the second example drawing (figure 3 of 3). The ED pipe is capable of processing challenging EDs with various artifacts, such as missing lines between arrowheads. This example contains only a single view, the third external dimension is determined by the diameter 30Ø. ED source Wiener Linien [wl].

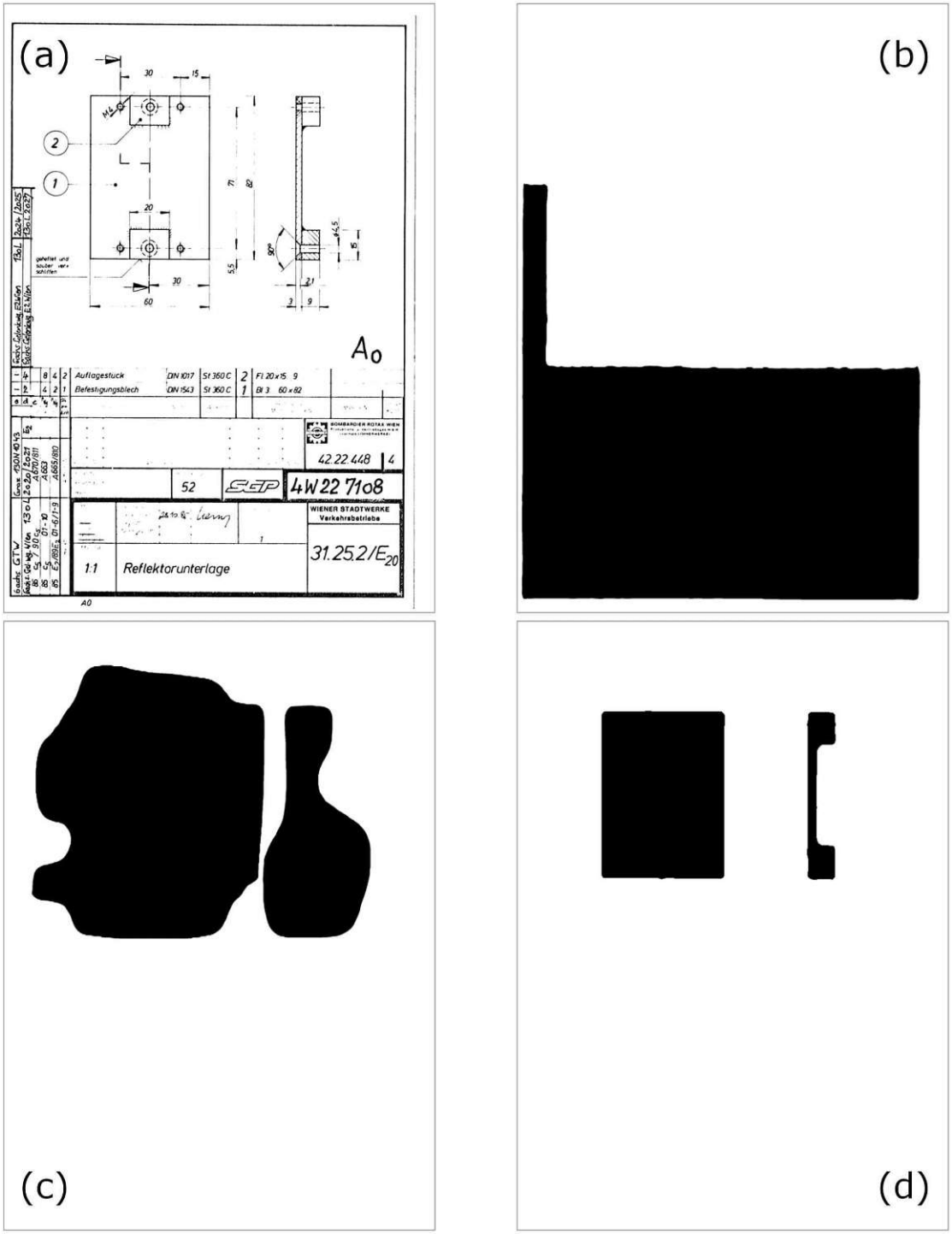


Figure 5.9: End-to-end processing of the third example drawing (figure 1 of 4). (a) Input image; (b) text-field segmentation result; (c) view segmentation result; (d) object view segmentation result. ED source Wiener Linien [wl].

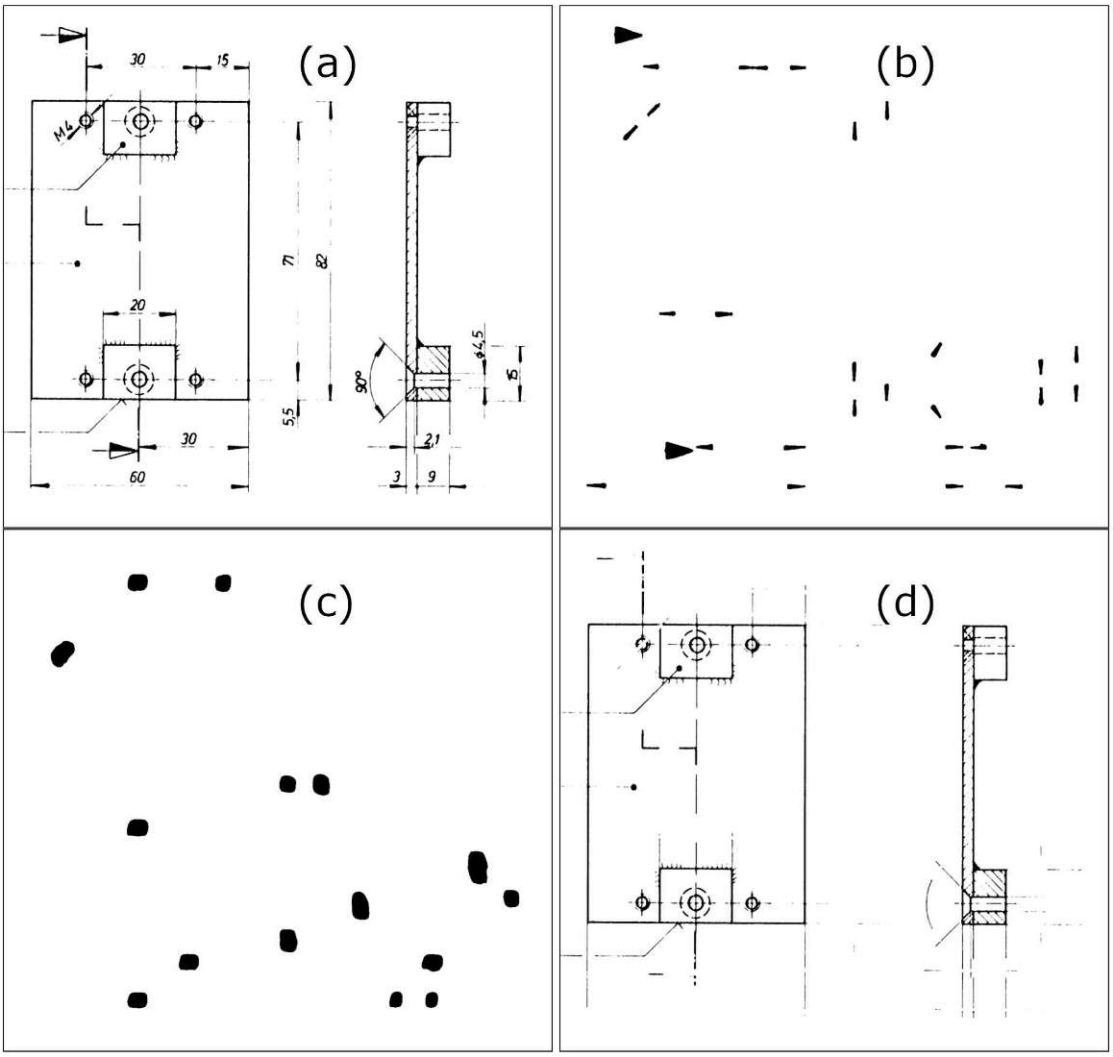


Figure 5.10: End-to-end processing of the third example drawing (figure 2 of 4). (a) Input image crop; (b) arrowhead segmentation result; (c) dimension number segmentation result; (d) simplified input image for object view segmentation. ED source Wiener Linien [wl].

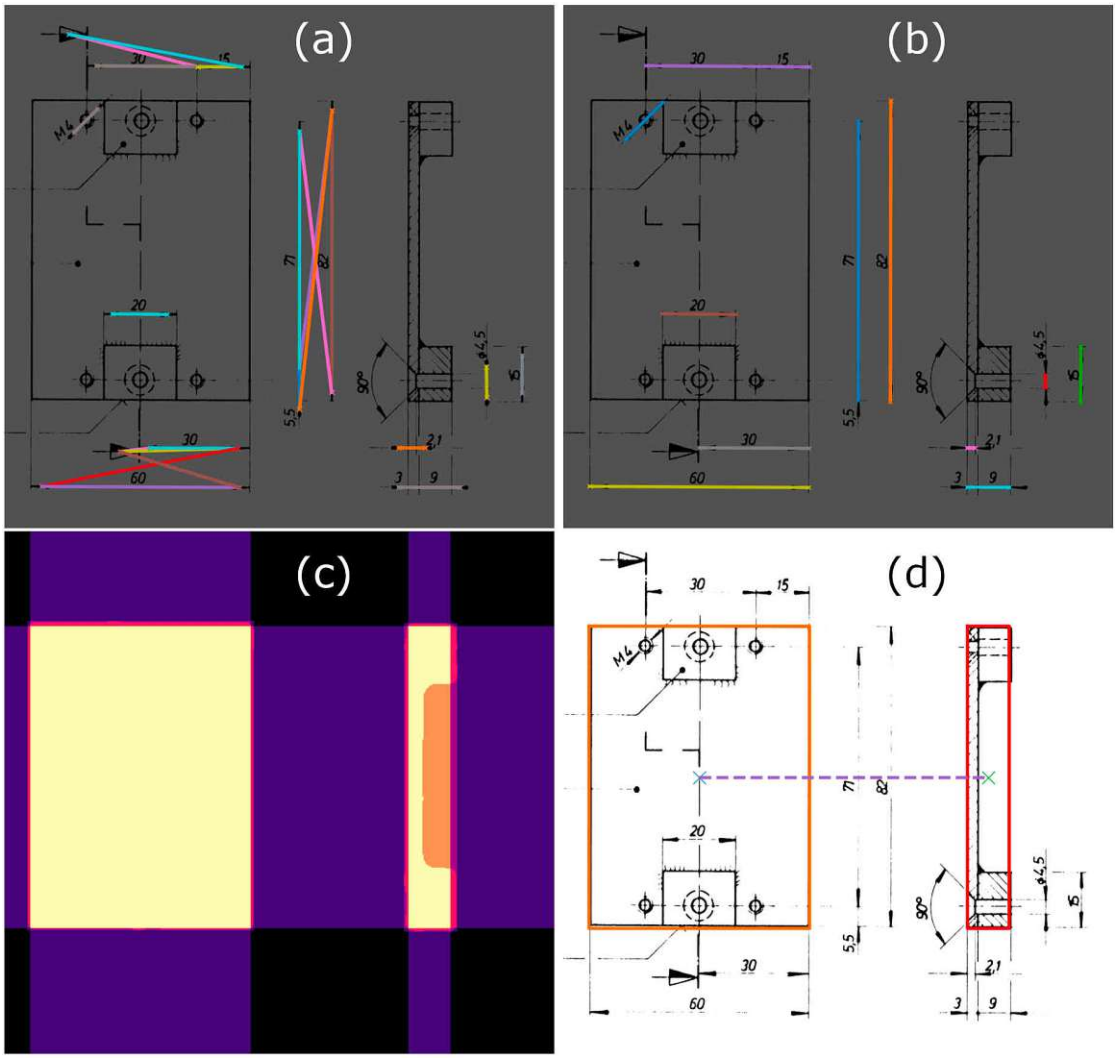
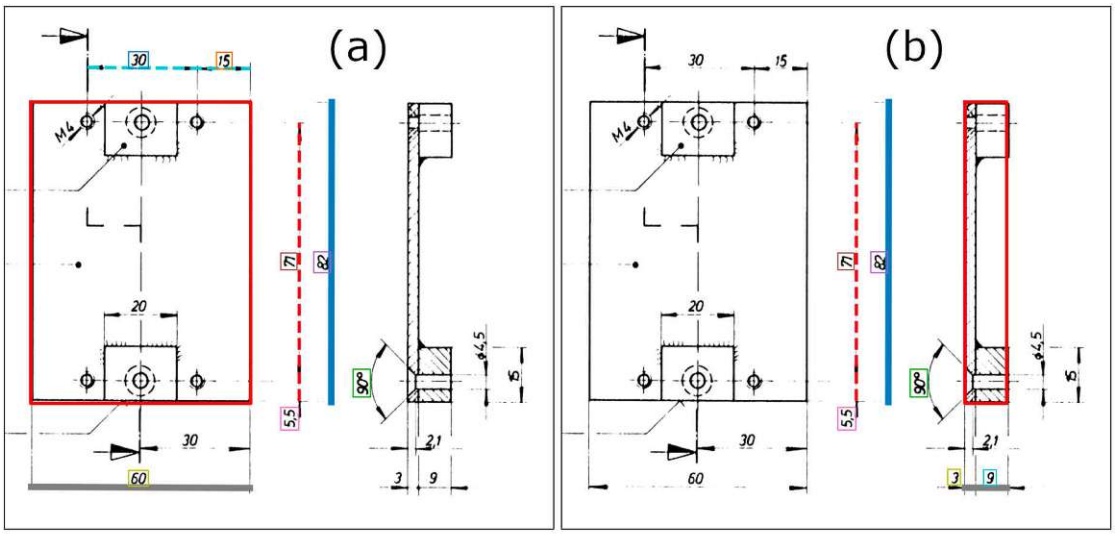


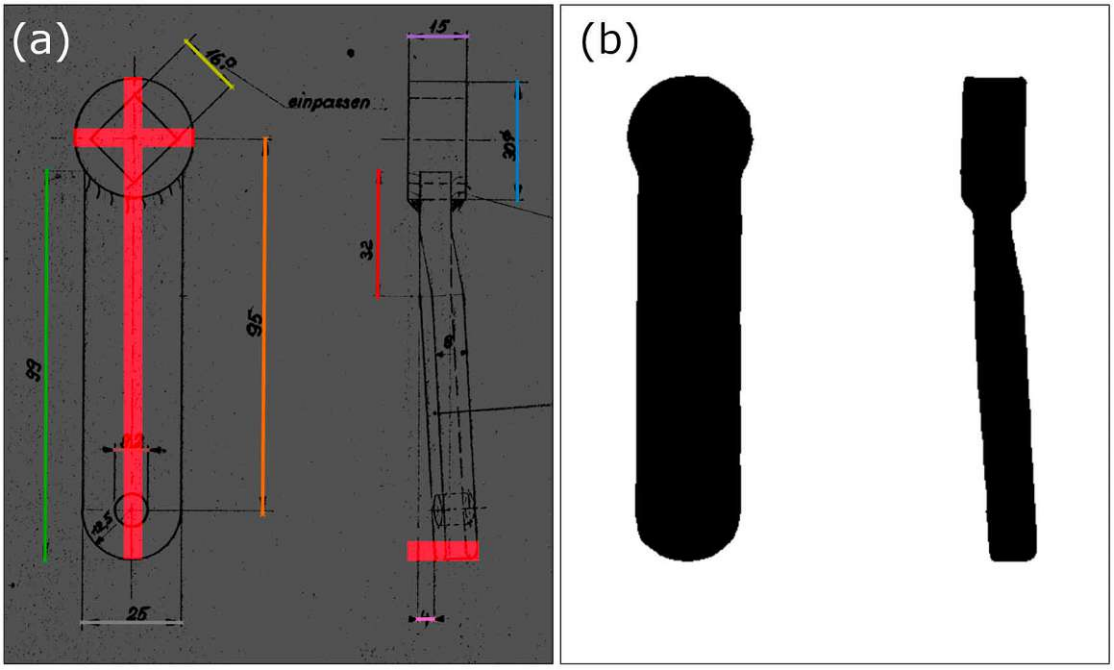
Figure 5.11: End-to-end processing of the third example drawing (figure 3 of 4). (a)(b) Dimension line filtering; (c)(d) view grid map and adjacency of grid cells. ED source Wiener Linien [wl].



(c)

```
{
    "line_x": 59.76,
    "line_y": 82.0,
    "line_z": 11.86,
    ...
}
```

Figure 5.12: End-to-end processing of the third example drawing (figure 4 of 4). Assignment of axis labels and associated dimension lines to each detected view in the grid to output the external dimensions. In this example, the third external dimension is measured by a chained dimension (bottom-right in (b), 3 mm + 9 mm = 12 mm). Our algorithm is able to produce an approximation of 11.865 mm, which is close to the true length of 12 mm. ED source Wiener Linien [wl].



(c)

```
{
    "line_x": 25.17,
    "line_y": 99.0,
    "line_z": 14.90,
    "object_x": 32.45,
    "object_y": 123.51,
    "object_z": 19.74,
    ...
}
```

Figure 5.13: An example of an object that is not fully measured by the dimension lines alone. The ED pipe produces a sufficiently accurate object view segmentation that allows us to determine the actual extent of the object in all three external dimensions drawn as thick red lines in (a). ED source Wiener Linien [wl].

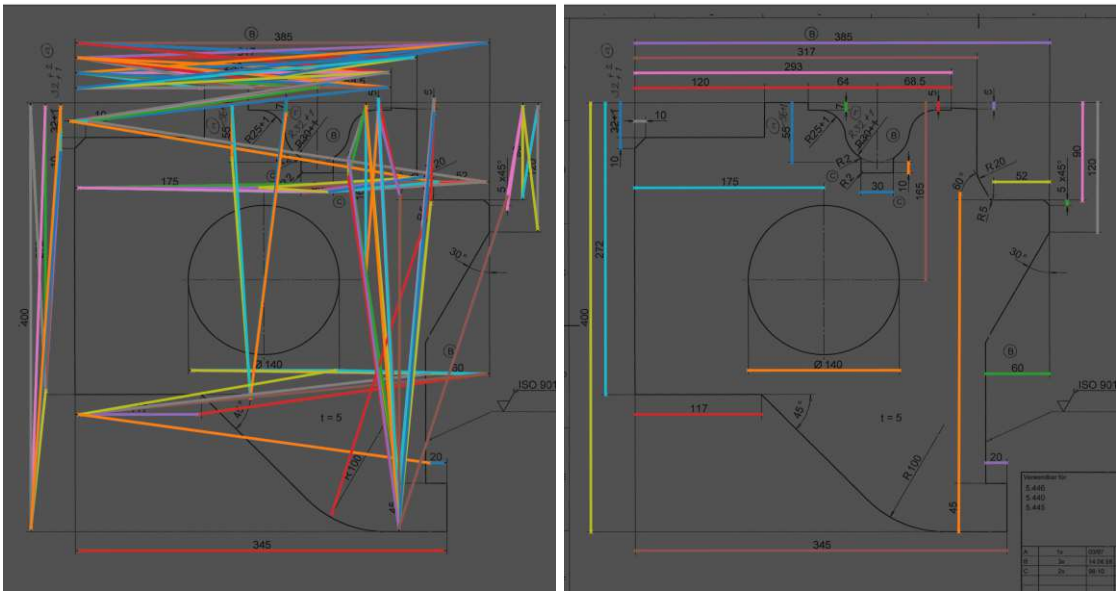


Figure 5.14: Dimension line processing example in a complex ED. ED source Wiener Linien [wl].

algorithm successfully detects the individual views and correctly discards the unwinding from the main grid. However, it erroneously included the isometric view in the main grid. Tighter tolerances when computing the adjacency matrices would alleviate this issue, but would compromise the generalizability of the algorithm. Figure 5.16 shows an example of a hand-drawn ED where the information is quite cluttered. The pipeline delivers good results, but we can see in the right image that the dimension line processing falsely merged two sets of arrowheads, which we highlighted in orange. Finally, Figure 5.17 shows a challenging example in (a) and a drawing containing only chained dimensions in (b). The challenging drawing in (a) works well, but our algorithm missed one dimension line on the far left of the image frame. The algorithm identifies the chained dimensions in (b) correctly but has some difficulty in finding the starting point of the horizontal lines.

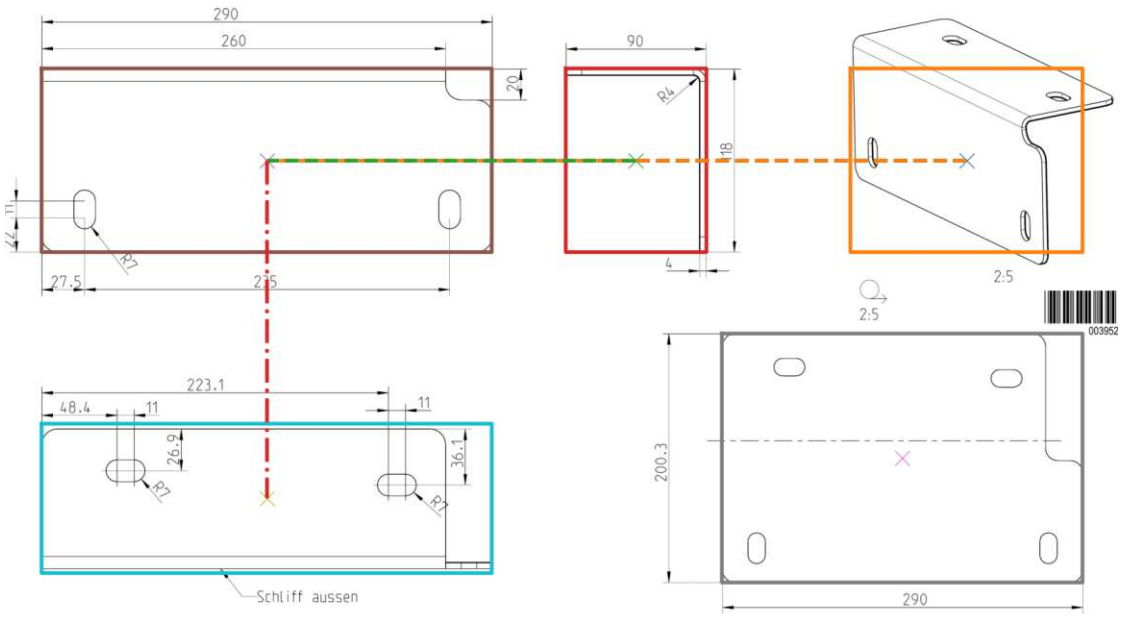


Figure 5.15: Example of a larger view grid. The algorithm correctly excludes the unwinding view in the bottom-right of the frame but falsely includes the isometric view. ED source Wiener Linien [wl].

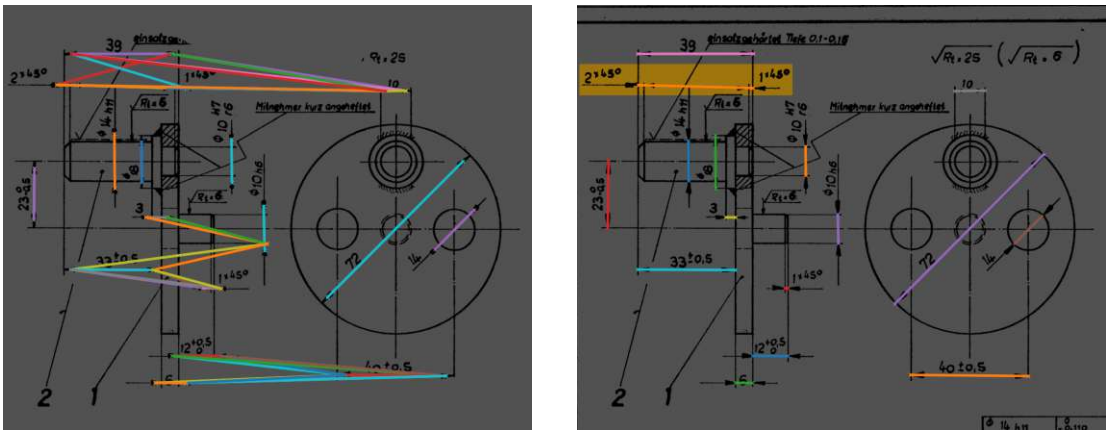


Figure 5.16: Dimension line processing of an ED that contains visual clutter. Note the false positive dimension line in the right image after processing highlighted in the orange box. It connects two sets of arrowheads between the dimension numbers $2^{\times 45°}$ and $1^{\times 45°}$. ED source Wiener Linien [wl].

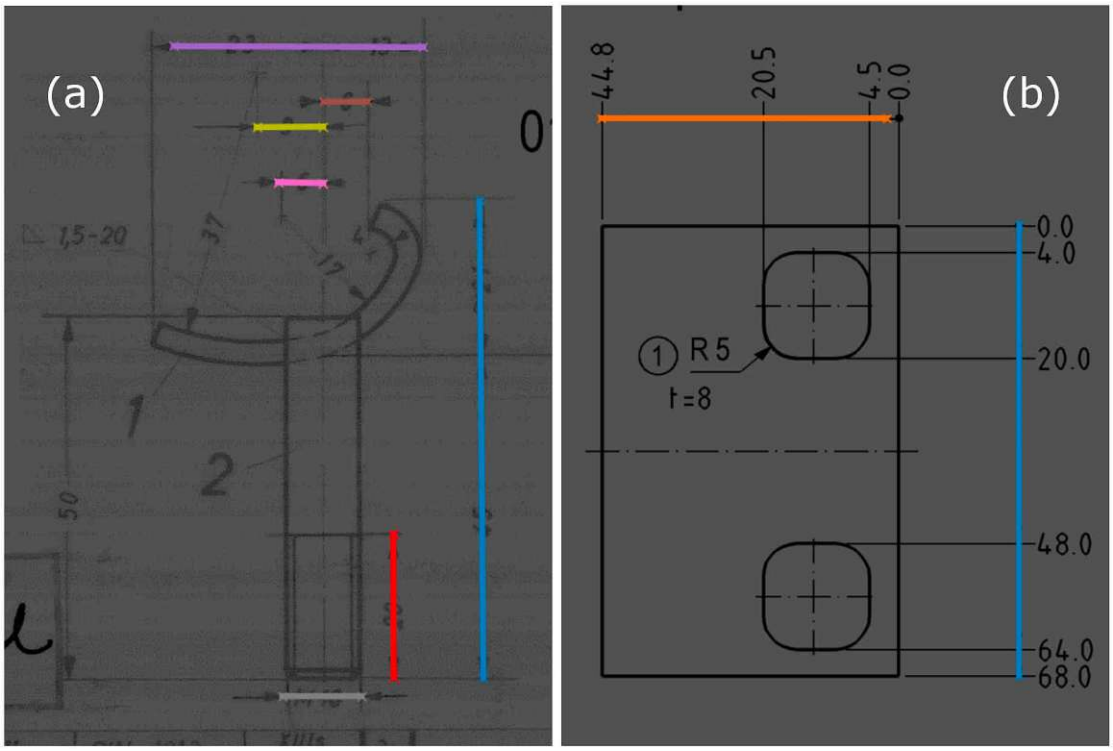


Figure 5.17: The algorithm missed one dimension line in the challenging example (a). The example in (b) confirms that our ED pipe is capable of processing EDs that only contain chained dimensions. ED source Wiener Linien [wl].

CHAPTER 6

Discussion

In this chapter, we reflect on our work and present possible improvements. We start with the proposed methodology and discuss our results. In addition, we address shortcomings of our approach and how we can potentially alleviate them. Finally, we analyze our prototypical implementation and explore ways to enhance its efficiency.

6.1 Methodology and Results

Our results indicate that the proposed approach is capable of automatically extracting external dimensions from diverse EDs and confirm that the feature selection and pipeline design are suitable to solve this problem. Our method works with various drawing styles and is able to reliably extract abstract information and important entities from real-world EDs. We design a modular and interpretable processing method in which individual stages could be replaced with optimized algorithms in the future, without breaking the entire system. AI algorithms play a critical role in our pipeline, but we try to restrict them to specific, well-defined tasks and do not opt for an end-to-end black-box solution. This strategy gives users greater flexibility in selecting tolerances and other hyperparameters to suit their particular data distribution. For example, if users want to process CAD drawings exclusively, they can set much smaller tolerance values.

In our analysis of ED characteristics, we find that arrowheads are one of the most important and robust features in drawings of all time periods, which is why they are central to our processing method. Their typically distinctive shape makes them a prime target for training a segmentation or detection algorithm. Furthermore, they are embedded in a rich context, because ED standards impose strict rules on their positioning within a drawing. We find that lines are essential components of EDs, but are not reliable enough to form the basis of our methodology. If we were working with modern drawings only, lines would be the main detection target. However, we find multiple problems with lines in older drawings, such as indistinguishable line types and missing lines due to

drawing inaccuracies or scan artifacts. AM domain experts are interested in the contents of the title block, which is why we decide to extract textfields from the input drawing. Our research partners will implement a system to extract relevant information from the detected textfields. We find that contextual information is crucial when trying to extract abstract complexity measures from EDs, which is also suggested in the related literature [ZJY+23]. Therefore, we decide to recognize dimension numbers, surface finish symbols, views, and object views. In our ED analysis, we only consider drawings that adhere to the international ISO standard because we do not have access to ASME data for training. It should be noted that ISO and ASME share many similarities, which should allow our pipeline to produce acceptable results even on ASME drawings, but we did not evaluate this use case. Through our analysis, we also find that the generation of synthetic examples is possible for many ED entities, allowing us to train robust segmentation models on a varied dataset. This finding is supported in the related literature [SNH23] [ZCK+22].

There is a lack of openly available data sets in this domain, as companies want to protect their sensitive information and intellectual property. Regrettably, our contributions cannot include the publication of an open data set for the aforementioned reasons. We have received data from our partners Wiener Linen and ÖBB, both for training and testing. These drawings were not annotated, so we spent a significant amount of time preparing the training and evaluation data. Our data set has an acceptable size, but critically, it contains a wide variety of different drawing styles from various epochs. We also have EDs with poor image quality and prominent artifacts. Although having a larger data set would be advantageous, it would only be valuable if the extra data exhibited a significant amount of diversity as well. Our data set contains only ISO drawings, so the inclusion of ASME examples would be interesting. The data set was enriched with the use of online data augmentation and, notably, the generation of synthetic samples. These methods allow us to train robust segmentation models on a relatively small amount of original data and improve the generalizability of the entire pipeline. Our synthetic data generators in combination with data augmentation provide us with a virtually unlimited number of training examples. Due to time constraints, we keep the generators simple, but a more sophisticated approach would be possible by leveraging the scripting capabilities of CAD software suites. A large language model (LLM) could generate specifications in text form to guide this process of generating synthetic drawings.

The proposed pipeline includes four image segmentation models. All of them achieve sufficient accuracy to locate the respective entities. Furthermore, due to our data generation and augmentation strategy, the models seem to produce robust results when confronted with unseen data. We note that there may be bias in our test data set because it was drawn from the same data sources as the training data set. Nevertheless, we are confident that our models will generalize to other examples from different sources. Precise arrowhead segmentation is critical because it allows us to determine the arrowhead orientation, which would not be possible with simple axis-aligned bounding boxes. Knowing the orientation is necessary for finding dimension lines and for linking arrowheads

to object views. Our arrowhead segmentation is accurate enough so that we can reliably calculate orientations. The combination of arrowhead, dimension number, and symbol segmentation in a single model makes sense because it provides the model with additional context to form its prediction. We also conducted experiments with annotated line types in addition to the aforementioned classes, but the results did not improve significantly. Panoptic segmentation [KHG+19] would enable us to segment all classes in one pass using the same model. An advantage of multiple segmentation models, as proposed in our work, is the ability to use different-sized inputs with varying context, for example, small crops from a large image for arrowhead segmentation and a downsampled version of the full original image for textfield segmentation. The architecture we selected is resource intensive, so it would be interesting to compare its performance to other, more lightweight architectures, which we could not evaluate due to time constraints. We performed experiments with multiple U-Net variants and find that the qualitative performance of the transformer-based architecture is superior. Multiple related works perform line thinning as part of their image preprocessing strategy [XLF+22] [ABF+98] [ZJY+23], which could be an interesting experiment for us. Furthermore, it would be valuable to experiment with GNNs to segment entities in old EDs, as most of the related work was done with contemporary data.

Filtering potential dimension lines is a key component of our pipeline and is based on a series of heuristics based on discussions with domain experts and on findings from our analysis of ED characteristics. Overall, we are satisfied with the performance of these filtering steps. The approach is currently limited to processing linear dimension sets only. Dealing with angular measurements and dimension sets linked to a single arrowhead would be an obvious improvement. The filtering algorithm is capable of untangling clusters of potential dimension lines in most cases. We could utilize the existing graph representation of these lines and their relations to infer a GNN for eliminating false positives. A limitation of the current approach lies in the construction of the view grid map because the boxes are axis-aligned and might not be representative in skewed drawings or EDs scanned at an angle. This could be alleviated by rectifying the input image after dimension line filtering. Despite this limitation, we are satisfied with the performance of our grid identification algorithm.

An advantage of the ED pipe is that OCR is performed near the end, so the preceding steps are not affected by potential errors in the OCR results. In fact, if the input ED is drawn on a $1:1$ scale and dots per inch (DPI) is stored in the image metadata, we can infer the correct dimensions without the need for OCR. This would also be possible if we could detect scale indications in EDs and link them to individual views. We observe that conventional OCR solutions have difficulty producing a reliable output when faced with image crops of EDs. My colleague Lisa Maria Kellner implemented a synthetic data generator and fine-tuned a detection and a recognition model, which we use in our prototypical pipeline implementation. The adapted OCR system works well on our diverse data set, but is still not capable of correctly recognizing text when rotated. To overcome this common problem of OCR methods, we predict the text of dimension

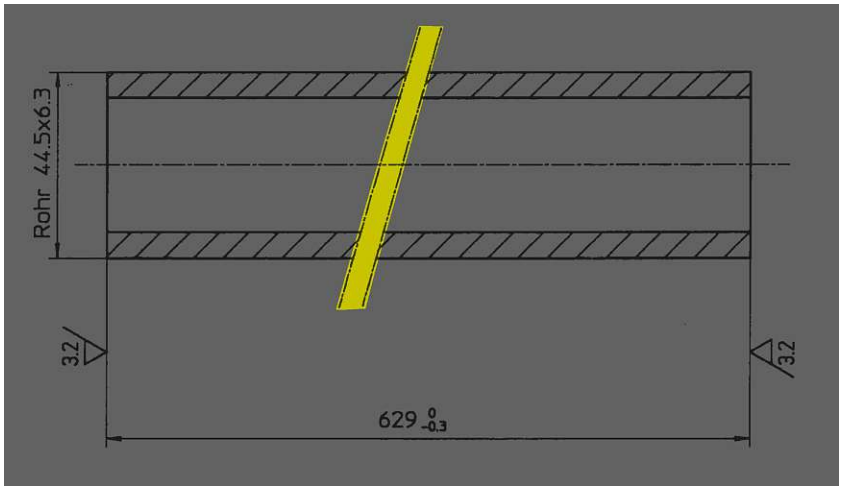


Figure 6.1: Detection of break lines in object views. ED source Wiener Linien [wl].

numbers in four 90° rotations and find an optimal OCR-to-pixel ratio that improves the resilience to errors in text recognition. A benefit of this strategy is that the pipeline is capable of determining the external dimensions of objects that are not fully specified by their dimension sets alone. This often occurs if the depicted object contains standardized components that are determined by DIN numbers. Another interesting approach to extract information from documents without an OCR system is the DONUT model, introduced by Kirillov et al. [KHY+22], which we could not try due to time constraints. A special case that our pipeline does not handle is object views with break lines. We experimented with break line detection and first results looked promising, as can be seen in Figure 6.1, but the full integration into our method is beyond the scope of this thesis.

Finally, we could integrate the following improvements into the ED pipe:

- Deal with angular measurements
- Handle single arrowhead dimensions
- Learn dimension line processing with a GNN
- Handle break lines
- Extract additional complexity measures, such as curvature, number of drill holes, bending part classification, symbol classification, volume, tolerances

Volume is a key complexity measure to assess material costs for the additive manufacturing of an object. A naive volume estimation can be derived from the external dimensions by simply multiplying $X \times Y \times Z$. However, this is typically a severe overestimation of the actual volume of an object. We conducted experiments that involved reconstructing a volumetric representation of the 3D object using results generated by the ED pipe, and the initial results were encouraging, as can be seen in Figure 6.2.

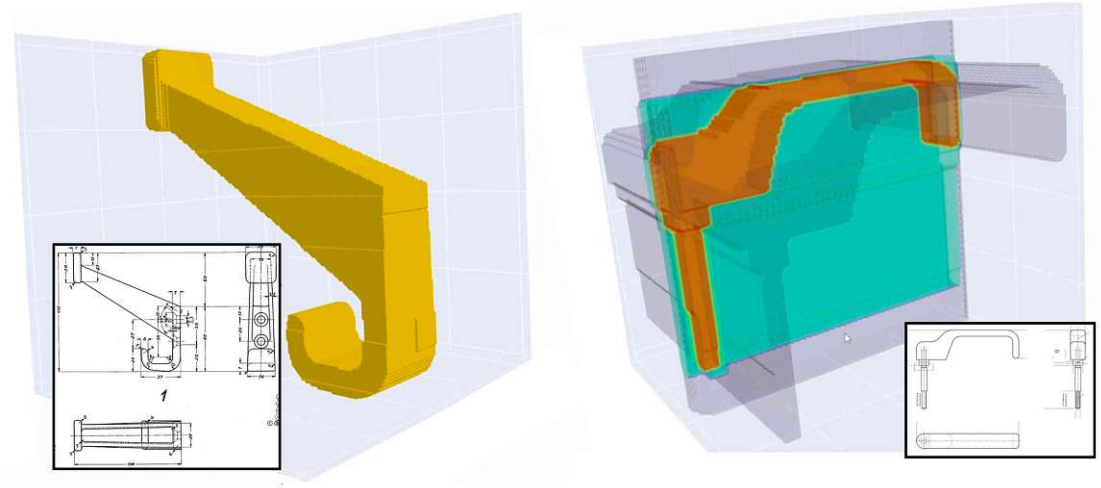


Figure 6.2: Experimental volumetric 3D reconstruction results of objects depicted in EDs. ED source Wiener Linien [wl].

6.2 Implementation

In this thesis, we implement a functional prototype of the proposed ED pipe that our partners will use in the AM4Rail research project [fhaa]. The implementation serves as a proof-of-concept and is not ready for commercial use. Wiener Linien and ÖBB will utilize the application to analyze several thousand EDs from their databases and will share feedback with us, which will be valuable in improving the application in the future. The prototype executes the individual pipeline stages in sequence. This simplifies the implementation, but does not utilize the computing resources to their full potential. Moreover, our implementation is designed to run locally on a single machine. Leveraging cloud computing and parallelization of pipeline stages would lead to increased runtime performance. Currently, an ED of average complexity takes about ten to fifteen seconds to process on the hardware described in Section 4.3. All segmentation stages could be executed in parallel, which would significantly reduce processing time. However, this engineering task is beyond the scope of this thesis. Overall, we are satisfied with the technology stack based on Python and PyTorch. It provides us with a comprehensive set of tools to implement, train, and run the pipeline without excessive software engineering overhead. Furthermore, the chosen libraries work well together in the NumPy ecosystem, which simplifies all tasks that involve image processing and tensors. Python supports many programming paradigms, which gives us the flexibility to write the implementation mostly in a functional style. Although it is an interpreted language and has a reputation for being slow, we find that this is not a limitation if using dedicated libraries such as OpenCV, PyTorch, SciPy, or NumPy and their respective vector operations whenever possible. Finally, PyTorch-Lightning is an invaluable tool for tracking experiments and saving the corresponding configurations.

CHAPTER 7

Conclusion and Future Work

In this thesis, we have developed and evaluated a method suitable for the fully automatic extraction of complexity measures from EDs, specifically focusing on the external dimensions of objects along the three main axes. These complexity measures serve as the basis for an AM potential analysis conducted by our research partners. The approach has demonstrated promising end-to-end results, highlighting strong generalizability across various types of EDs. In addition, the intermediate results of the individual pipeline stages have consistently achieved good metrics, confirming the robustness of the approach. Synthetic data generation played an important role in the training of robust models. The proposed method is based on a handful of critical ED entities, including arrowheads, dimension numbers, textfields, views, and object views, which we identified in an in-depth analysis of drawings from different time periods and of varying visual quality. A distinctive feature of the ED pipe is its resilience to OCR errors, achieved by calculating an optimized ratio between the OCR results and the length of dimension lines in pixels. This strategy ensures the reliability of the extracted external dimensions even in the presence of some textual inaccuracies. Moreover, our pipeline is designed to be interpretable and modular, facilitating easy modifications and updates. We implemented a functional prototype of the system that is currently in use by our partners in the AM4Rail research project [fhaa]. The method was designed and evaluated with EDs from the European rail industry that adhere to the ISO standard without considering the specific features of other standards or drawings from different industries. The pipeline is limited to detecting linear dimension sets and cannot handle angular measurements. A limitation of optimizing the OCR-to-pixel ratio are object views that contain break lines, which we currently cannot process correctly. We believe that additional features could be added to the pipeline without too much effort, including the handling of angles, break lines, and the extraction of other complexity measures.

There is substantial potential for further development of our method to enhance its capabilities and applicability. Key areas of future work include the following:

- **Recognition of Additional Complexity Measures:** Expanding our method to identify and quantify additional complexity measures such as volume, curvature, the number of drill holes, and bending part classification.
- **Learning Dimension Line Processing with GNNs:** Employing a GNN for the processing of dimension lines based on the graph representation we use in our pipeline. Related work has shown that GNNs are suitable for this task [ZJY+23] [XLF+22].
- **Full 3D Reconstruction:** Develop methods for automatic conversion from 2D drawings to 3D CAD models. Results obtained from our ED pipe could guide the reconstruction process.

List of Figures

Acronyms

AI artificial intelligence. 3, 67, 68, 95

AM additive manufacturing. 1–3, 5, 20, 33, 43, 63, 65, 67, 96, 101

CAD computer-aided design. 1, 3, 5, 6, 8, 10, 19, 20, 28, 29, 31, 34, 40, 55, 77, 95, 96, 102

CNN convolutional neural networks. 41, 42

DL deep learning. 5, 36, 38, 68, 69

DPI dots per inch. 97

DSC Dice similarity coefficient. 45, 75, 76

ED engineering drawing. 1–8, 10–14, 16, 17, 19, 20, 22, 24–31, 33–36, 38–41, 43, 44, 46–49, 52–55, 58–65, 67–69, 71–99, 101–107

FPN feature pyramid network. 41–44, 47, 50, 53, 71

GCN graph convolutional network. 5, 6, 10

GNN graph neural network. 8, 10, 97, 98, 102, 103

LLM large language model. 96

LR learning rate. 46, 47, 50, 53, 71

MiT mix transformer encoder. 42–44, 47, 50, 53, 71, 72

NN neural networks. 36

OCR optical character recognition. 3, 7, 8, 10, 11, 33, 34, 36, 61–64, 69, 76, 77, 82, 83, 97, 98, 101, 106

PCA principal component analysis. 54, 68

PDF portable document format. 69

PPE percentage prediction error. 76, 82

SVM support vector machine. 8, 10, 103

ViT vision transformer. 43

Bibliography

[ABF+98] S Ablameyko, V Bereishik, O Frantskevich, M Homenko, and N Paramonova. A System for Automatic Recognition of Engineering Drawing Entities. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, volume 2, pages 1157–1159 vol.2, 1998.

[alb] Albumentations. `https://albumentations.ai`. Accessed: 2024-03-30.

[aut] Auto-py-to-exe. `https://github.com/brentvollebregt/auto-py-to-exe`. Accessed: 2024-03-31.

[Cho54] G Choquet. Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295, 1954.

[cud] CUDA. `https://developer.nvidia.com/cuda-zone`. Accessed: 2024-03-30.

[DP73] DH Douglas and TK Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[dxf] Adobe DXF File Format. `https://www.adobe.com/at/creativecloud/file-types/image/vector/dxf-file.html#`. Accessed: 2024-04-18.

[EKS+96] M Ester, HP Kriegel, J Sander, X Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[ffg] Österreichische Forschungsförderungsgesellschaft FFG. `https://www.ffg.at/en`. Accessed: 2024-04-19.

[fhaa] Fraunhofer Austria. `https://www.fraunhofer.at/de/presse/news/am4rail-nov-2021.html`. Accessed: 2024-04-19.

[fhab] Fraunhofer IAPT. https://www.iapt.fraunhofer.de. Accessed: 2024-04-19.

[fon] Google Fonts. https://fonts.google.com. Accessed: 2024-03-25.

[fra] Engineering Drawing Basics. https://fractory.com/engineering-drawing-basics/. Accessed: 2024-03-27.

[GBF95] W Guang, M Baraldo, and M Furlanut. Calculating Percentage Prediction Error: A User's Note. *Pharmacological Research*, 32(4):241–248, 1995.

[ham] Hampel Filter Python Package. https://github.com/MichaelisTrofficus/hampel_filter. Accessed: 2024-03-30.

[HWC+23] K Han, Y Wang, H Chen, X Chen, J Guo, Z Liu, Y Tang, A Xiao, C Xu, Y Xu, Z Yang, Y Zhang, and D Tao. A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):87–110, 2023.

[HYL17] W Hamilton, Z Ying, and J Leskovec. Inductive Representation Learning on Large Graphs. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[IKCM16] AM Intwala, K Kharade, R Chaugule, and A Magikar. Dimensional Arrow Detection from CAD Drawings. *Indian Journal of Science and Technology*, 9(21), 2016.

[KHG+19] A Kirillov, K He, R Girshick, C Rother, and P Dollár. Panoptic Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9404–9413, 2019.

[KHY+22] G Kim, T Hong, M Yim, J Nam, J Park, J Yim, W Hwang, S Yun, D Han, and S Park. OCR-Free Document Understanding Transformer. In S Avidan, G Brostow, M Cissé, GM Farinella, and T Hassner, editors, *Computer Vision – ECCV 2022*, pages 498–517, Cham, 2022. Springer Nature Switzerland.

[KSL+21] Z Kuang, H Sun, Z Li, X Yue, TH Lin, J Chen, H Wei, Y Zhu, T Gao, W Zhang, K Chen, W Zhang, and D Lin. MMOCR: A Comprehensive Toolbox for Text Detection, Recognition and Understanding. *arXiv preprint arXiv:2108.06543*, 2021.

[LDG+17] TY Lin, P Dollar, R Girshick, K He, B Hariharan, and S Belongie. Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[LH17] I Loshchilov and F Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[lig] PyTorch-Lightning. https://lightning.ai/pytorch-lightning. Accessed: 2024-03-30.

[LQXL21] X Li, W Qian, D Xu, and C Liu. Image Segmentation Based on Improved Unet. *Journal of Physics: Conference Series*, 1815(1):012018, February 2021.

[LTH+23] YH Lin, YH Ting, YC Huang, KL Cheng, and WR Jong. Integration of Deep Learning for Automatic Recognition of 2D Engineering Drawings. *Machines*, 11(8):802, 2023.

[MBP+22] S Minaee, Y Boykov, F Porikli, A Plaza, N Kehtarnavaz, and D Terzopoulos. Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022.

[mcga] McGill University - Basics of Graphics Communication. https://www.mcgill.ca/engineeringdesign/step-step-design-process/basics-graphics-communication. Accessed: 2024-03-27.

[mcgb] View Axis Labels in Engineering Drawing. https://www.mcgill.ca/engineeringdesign/files/engineeringdesign/styles/wysiwyg_extra_large/public/3_projections.jpg?itok=GWcAQL2z. Accessed: 2024-03-25.

[mcgc] View Grid Structure in Engineering Drawing. https://www.mcgill.ca/engineeringdesign/files/engineeringdesign/styles/wysiwyg_extra_large/public/1st_angle_projection-m_-_220815.jpg. Accessed: 2024-03-24.

[net] NetworkX - Network analysis in Python. https://networkx.org. Accessed: 2024-03-31.

[new] Basic Elements of Engineering Drawings. https://newtonianworld.com/mechanical-design-engineering-topics/engineering-drawings/basic-elements-of-engineering-drawings/. Accessed: 2024-03-27.

[num] NumPy. https://numpy.org. Accessed: 2024-03-30.

[ocw] Design Handbook: Engineering Drawing and Sketching. https://ocw.mit.edu/courses/2-007-design-and-manufacturing-i-spring-2009/pages/related-resources/drawing_and_sketching/. Accessed: 2024-03-27.

[oeb] Österreichische Bundesbahnen. https://www.oebb.at. Accessed: 2024-04-19.

[ope] OpenCV Computer Vision Library. https://opencv.org. Accessed: 2024-03-30.

[Ots79] N Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[pdf] PDF2Image Python Package. https://github.com/Belval/pdf2image. Accessed: 2024-03-31.

[PGK+20] D Prasad, A Gadpal, K Kapadni, M Visave, and K Sultanpure. CascadeTabNet: An Approach for End to End Table Detection and Structure Recognition From Image-Based Documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[PNAG16] RK Pearson, Y Neuvo, J Astola, and M Gabbouj. Generalized Hampel Filters. *EURASIP Journal on Advances in Signal Processing*, 2016(1), August 2016.

[pyi] PyInstaller Documentation. https://pyinstaller.org/en/stable/. Accessed: 2024-03-31.

[pyta] Python Programming Language. https://www.python.org. Accessed: 2024-03-30.

[pytb] PyTorch. https://pytorch.org. Accessed: 2024-03-30.

[pytc] PyTorch NumPy Bridge. https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html#bridge-to-np-label. Accessed: 2024-03-30.

[RDS+15] O Russakovsky, J Deng, H Su, J Krause, S Satheesh, S Ma, Z Huang, A Karpathy, A Khosla, M Bernstein, AC Berg, and L Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RFB15] O Ronneberger, P Fischer, and T Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In N Navab, J Hornegger, WM Wells, and AF Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[scia] Scikit-Image. https://scikit-image.org. Accessed: 2024-03-30.

[scib] SciPy. https://scipy.org. Accessed: 2024-03-30.

[seg] Segmentation Models PyTorch. https://github.com/qubvel/segmentation_models.pytorch. Accessed: 2024-03-30.

[sha] Shapely Python Library Documentation. https://shapely.readthedocs.io/en/stable/manual.html. Accessed: 2024-03-30.

[SHL+24] Z Song, Z He, X Li, Q Ma, R Ming, Z Mao, H Pei, L Peng, J Hu, D Yao, and Y Zhang. Synthetic Datasets for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Vehicles*, 9(1):1847–1864, 2024.

[skl] Scikit-Learn. https://scikit-learn.org/stable/. Accessed: 2024-03-31.

[SMRM21] B Scheibel, J Mangler, and S Rinderle-Ma. Extraction of Dimension Requirements from Engineering Drawings for Supporting Quality Control in Production Processes. *Computers in Industry*, 129:103442, 2021.

[SNH23] T Schlagenhauf, M Netzer, and J Hillinger. Text Detection on Technical Drawings for the Digitization of Brown-Field Processes. *Procedia CIRP*, 118:372–377, 2023.

[sur] Surface Finish Symbols. https://www.keyence.com/Images/ss_microscope_roughness_line_img_flow_01_1578335.jpg. Accessed: 2024-03-27.

[SY22] Z Song and H Yao. Segmentation Method of U-net Sheet Metal Engineering Drawing Based on CBAM Attention Mechanism. *arXiv preprint arXiv:2209.14102*, 2022.

[ten] TensorBoard. https://www.tensorflow.org/tensorboard. Accessed: 2024-03-30.

[tes] Tesseract OCR. https://github.com/tesseract-ocr/tesseract. Accessed: 2024-03-24.

[tif] Tifffile Python Library. https://github.com/cgohlke/tifffile/. Accessed: 2024-03-31.

[tum] Grundlagen des Technischen Zeichnens, TU Munich. https://www.ei.tum.de/fileadmin/tueifei/ewt/Werkstatt/Zeichnungserstellung.pdf. Accessed: 2024-03-27.

[Vap00] VN Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, 2000.

[vrv] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH. https://www.vrvis.at. Accessed: 2024-04-19.

[VSP+17] A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, AN Gomez, L Kaiser, and I Polosukhin. Attention is All you Need. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[w24] Title Block. `https://docs.werk24.io/features/title_block.png`. Accessed: 2024-03-27.

[WBL23] CY Wang, A Bochkovskiy, and HYM Liao. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7464–7475, June 2023.

[wl] Wiener Linien. `https://www.wienerlinien.at`. Accessed: 2024-04-19.

[WPLK18] S Woo, J Park, JY Lee, and IS Kweon. CBAM: Convolutional Block Attention Module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[WSL+19] Y Wang, Y Sun, Z Liu, SE Sarma, MM Bronstein, and JM Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 38(5):1–12, October 2019.

[WT04] L Wendling and SA Tabbone. A New Way to Detect Arrows in Line Drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):935–941, 2004.

[XLF+22] L Xie, Y Lu, T Furuhata, S Yamakawa, W Zhang, A Regmi, L Kara, and K Shimada. Graph Neural Network-Enabled Manufacturing Method Classification from Engineering Drawings. *Computers in Industry*, 142:103697, 2022.

[XWY+21] E Xie, W Wang, Z Yu, A Anandkumar, JM Alvarez, and P Luo. SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. In M Ranzato, A Beygelzimer, Y Dauphin, PS Liang, and J Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12077–12090. Curran Associates, Inc., 2021.

[YRN+22] M Yeung, L Rundo, Y Nan, E Sala, CB Schönlieb, and G Yang. Calibrating the Dice Loss to Handle Neural Network Overconfidence for Biomedical Image Segmentation. *Journal of Digital Imaging*, 36(2):739–752, December 2022.

[ZCK+22] W Zhang, Q Chen, C Koz, L Xie, A Regmi, S Yamakawa, T Furuhata, K Shimada, and LB Kara. Data Augmentation of Engineering Drawings for Data-Driven Component Segmentation. In *Volume 3A: 48th Design*

Automation Conference (DAC), International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 08 2022.

[ZJY+23] W Zhang, J Joseph, Y Yin, L Xie, T Furuhata, S Yamakawa, K Shimada, and LB Kara. Component Segmentation of Engineering Drawings using Graph Convolutional Networks. *Computers in Industry*, 147:103885, 2023.