The International Journal of Advanced Manufacturing Technology (2021) 115:2145–2159
https://doi.org/10.1007/s00170-021-07186-0

ORIGINAL ARTICLE

CAD integrated automatic recognition of weld paths

Tuan Anh Tran[1] · Andrei Lobov[1] · Tord Hansen Kaasa[1] · Morten Bjelland[2] · Ole Terje Midling[3]

Received: 7 January 2021 / Accepted: 29 April 2021 / Published online: 20 May 2021

Abstract
In this paper, a CAD integrated method is proposed for automatic recognition of potential weld locations in large assembly structures predominantly comprised of weld joints. The intention is to reduce the total man-hours spent on manually locating, assigning, and maintaining weld-related information throughout the product life cycle. The method utilizes spatial analysis of extracted stereolithographic data in combination with available CAD functions to determine whether the accessibility surrounding a given intersection edge is sufficient for welding. To demonstrate the method, a system is developed in Siemens NX using their NXOpen Python API. The paper presents the application of the method to real-life use cases in varying complexity in cooperation with industrial partners. The system is able to correctly recognize almost all weld lines for the parts considered within a few minutes. Some exceptions are known for particular intersection lines located deep within notched joints and geometries weldable through sequential assembly, which are left as a subject to further works.

Keywords Computer-aided design · Topology analysis · Feature recognition · Weld automation

1 Introduction

The demand for individualized and customized products calls for flexible and intelligent development processes. In such industries, a desire has risen to adopt robotic systems to automate the welding processes and further increase production efficiency. Automotive and aerospace industries have already successfully adopted many such methods for production of high volume products. However, when dealing with individualized products, several challenges arise with respect to manufacturing processes and product flexibility.

Systems created with knowledge-based engineering (KBE) aim to provide the flexibility and automation to the product development processes by capturing and characterizing engineering intent behind decisions. This captured knowledge, or logic, can be redeployed at a given time to automate decisions and design parameters, and provide smarter and more optimized solutions taking available information into account. However, in order to build such systems, methods for recognizing of specific features are considered a necessity [1, 2].

Industries dealing with individualized large metal structures where the predominant joint mechanism between parts is comprised of welds end up spending considerable amounts of time defining such weld joints. Furthermore, instructions in the form of 2D drawings, or robotic welding trajectory, have to be generated. When paired with a desire to provide customized products, these processes often need to be repeated due to changes in parts and geometry.

In this paper, we present an approach to automatically identify and assign potential welding paths based on geometric features in 2.5D. The feature recognition system is aimed to work in an integrated CAD environment to provide assistance to the development process, functioning as a cornerstone to KBE systems such as presented in

✉ Tuan Anh Tran
tuan.a.tran@ntnu.no

Andrei Lobov
andrei.lobov@ntnu.no

Tord Hansen Kaasa
tordkaasa@gmail.com

Morten Bjelland
morten.bjelland@leirvik.com

Ole Terje Midling
ole.terje.midling@m-a.no

[1] Norwegian University of Science and Technology, Richard Birkelands vei 2b, Trondheim, NO-7034, Norway

[2] Leirvik AS, Stord, Norway

[3] Marine Aluminium AS, Avaldsnes, Norway

Prescott et al. [3]. By integrating systems into a CAD environment, we can achieve high-level automation of tasks otherwise considered mundane yet time-consuming by humans with minimal disruption to existing workflows. The viability of the approach is investigated through an implementation in Siemens NX using the NXOpen Python API. Other well-established CAD systems also have or are developing their own respective APIs to allow users to expand on functionalities [4, 5]. Though, implementations on these systems have yet to be demonstrated.

The approach consists of several stages, starting with the extraction of intersecting curves between different solid bodies and assembly parts. Geometric features in the form of stereolithographic data are then extracted from the assembly model to be analyzed with respect to the topology. The intersecting curves are then used as a baseline for generating cross-sections of all the potential weld lines. The cross-sections can be generated by performing slices of the triangle-based geometry following arbitrary planes using intersection curve positions and direction. The generated cross-sections can be analyzed in a 2D environment using ray casting techniques to derive at values for accessibility and space from the weld joint. Furthermore, these spaces can be defined by controlling the length of the rays used, allowing non-accessible spaces to be filtered out based on the tool-size requirements. By focusing on the topology of the model and the cross-sections, the need for manual specification of and definition of parts is reduced while increasing the detection accuracy robustness.

In Section 3, the approach using Siemens NX and the NXOpen Python API for the CAD environment will be outlined in detail. As a final step, a demonstration of our solution following a number of use cases including those based on Leirvik AS and Marine Aluminum AS products, Fig. 8c and d respectively, is presented in Section 4. The system is able to successfully generate all correct weld lines depending solely on the geometrical data in the STEP files given.

2 Related work

Modern CAD systems come with Application Programming Interfaces (APIs) built-in. These kinds of API are particularly useful in supporting, and making it easier for Small and Medium Enterprises (SMEs) to develop their own customized and dedicated solutions. Repetitive work tasks commonly deemed mundane to humans can be replaced by lines of code, potentially saving thousands of work-hours down the line.

For instance, extension and use of the NXOpen API were described in [6]. The extended API allows a reduced amount of code needed from thousands to several lines making it more natural for human engineers to use. In [7], Visual Basic for Application (VBA) with Autodesk Inventor was used to implement an algorithm for machining feature (MF) recognition. The API allowed easier organization (a tree-like structure) and use of software objects to express and solve the feature recognition problem. In [8], integration of feature recognition approaches using Siemens NX and the NX Open API is investigated with the goal to accelerate production and manufacturing processes down the line.

Feature recognition systems play an important role in automating processes. CAD models can be analyzed in various ways to extract useful information of features for different applications. For example, to support welding information retrieval, topological analysis can be combined with rules in large assemblies to be able to automatically identify weld joints and types. This can be further extended to characterizing the joints types themselves to match strength and cost requirements. The recognition of machining features is in many cases considered a cornerstone to automatic machining [2]. Tool paths can be generated depending on various machining features. It is being sliced to recognize 2D cross-section/features. The importance of feature recognition systems was presented in [1]. The paper reports on a method of feature recognition using light rays from a given direction, i.e., machining direction. The method builds on a pre-existing paper [9].

The presence of APIs also allows customized KBE solutions to be integrated into the development framework. A KBE application for detecting manufacturing tasks on CAD models, which is used for recognizing and characterizing a few types of weld based on rule-based logic, was reported in [10]. The weld path is represented using ontologies as curve primitives: lines, circular arcs or B-spline curves. These can be fed to robot program planning systems without the need for complex geometry engines. The welding gaps could also be represented in the ontology. However, a persisting drawback is that the system only works by comparing two parts (plates) at the time. The information used to power the rules are wall direction, gap position, joint position, angle, throat thickness and the intersection. Furthermore, the scope did not consider more complex shapes where actual weldability and reachability might become a problem. Furthermore, in order to answer the demand in cost-efficiency the knowledge transfer between product design and process planning needs to become faster. Making the right selection of steps gives as a result low production costs and efficient results [11]. KBE can support automatic programming going for lot size one products [12].

CAD-based frameworks are developed to integrate information in CAD throughout the process of product development and manufacturing, exemplified in the automation of the coating process deriving programming solution for

robotic work cells [13]. An example case of integrated CAD methods to support manufacturing is reported in [14], where an algorithm to automatically detect open and closed spatial angles in edges is presented. Based on this specific knowledge about an edge feature, blends or chamfers could be assigned automatically. KBE approaches to assess weld types also exist as demonstrated in [15], where a CATIA Part file was used as an input to evaluate weldability and cost. MS Excel evaluation file was used with the implemented "IF-Then" logic to give an "ok", "not ok" output for specific weld methods, e.g., TIG or LaserBeam, in addition to the associated cost.

On the methods side, various approaches are used for features recognition. Among those is application of Convolutional Neural Network (CNN) on images of CAD models from specific angles [16]. In general, [17] outlines general means to improve the performance and training of neural networks. In subtractive manufacturing, a different approach based on generation of "slices" and analyzing through "light" rays to deal with visibility of surfaces [18]. Earlier research outlining methods used for state of the art geometric feature recognition techniques was published in 2001, presenting then "25 years of research in feature recognition" [19]. It is worth mentioning as it sets classification of feature recognition techniques for CAD models. One of the statements given in that paper is that "is fair to say that there is no single best technique for all types of features and applications" [19]. In other words, even today, 20 years later one may need to check for hybrid methods for feature detection.

Product design may include dealing with feature-based and non-feature objects [20]. The latter may refer to irregular shapes such as, for example, sculptures, which can be simply decomposed as "basic" or simple objects, e.g., holes and pockets. Still, such products and corresponding objects should be kept in mind as and if these should be integrated into more complex products and one check possibilities for automating analysis based on CAD drawings. This discussion brings to the manufacturability or design for manufacturability, where knowledge on manufacturing resources (e.g., equipment, CNC machines) could also affect product design phase.

Overall, being able to detect specific features as, for example, the holes, can be also important for the background of this research to check if potential welding lines are reachable (e.g., by a weld gun). In some cases CAD mesh simplification can be useful. With respect to this article, sometimes one can omit the features rather than finding those. Feature simplification can be done using FEA analysis, which can be used in order to simplify the geometry. This can include suppression of holes and pockets [21]. Also mesh-based hole filling and detection approach is discussed in [22]. Apparently, detection of holes in meshes is quite simple, because mesh nodes at the end are given as boundary seeds. The only thing that remains is to check connecting nodes and follow paths until a loop has been formed.

A particular challenge regarding the automated recognition and proposal of dedicated problem-specific additions (e.g., welds) based on the CAD drawings is also with the recognition of specific features in a part. For example, [23] proposed hole detection approach for triangular mesh models. As opposed to other methods, this method focuses on detecting holes in 3D geometry (triangular mesh models, STL, etc.) Another method for holes detection in mesh geometry with the presentation of boundary identification algorithm can be found in [24]. The algorithm considers different features of the hole such as upper, lower boundaries, and the filling. In other words, detection of a specific feature may require ad-hoc solutions with detailed representation of the feature. Also, reverse engineering approaches are known for reconstructing of 3D image from scanned features as reported in [25]. This is used for verification of drilled hole position. Complex holes recognition based on STL input using hybrid mesh segmentation is presented in [26]. However, holes are assumed having circular shapes only.

In [27], a method to automatically recognize spot welding features is presented. The method was integrated into CATIA, and relied on STEP files as input. Similarly to our findings, the lack of integrated automated welding feature recognition systems was noted despite the potential. Considering presented background knowledge, in this paper, we go towards more a generalizable weld-detection solution making it possible to analyze topology and geometry characteristics by a method joining slicing and ray casting using intersection curves used as a locational and orientational input to achieve a quantifiable weldability including that of accessibility of the potential welding lines.

3 Methodology

The proposed method in this paper is based on separate stages governing their own elements of analysis. The general idea is that cross-sections can be generated and further analyzed using a set of potential weld lines that have been derived from intersections between solid bodies inside a CAD environment. These 2D cross-sections can then be used as subject to further analysis by applying ray casting methods from the center of the weld to determine a form for point visibility. This is essentially the same as the methods [1, 18, 28], however, applied to a sliced 2D section for welds. Based on this point visibility, measured as a range of consecutive open space around the weld, the assumption is that a sense of weldability, or accessibility can

be determined. From this point, other rule-based inference methods can be used to filter out less optimal lines based on the angle of visibility from the weld joint to the outside accessible area, length and thickness of materials.

The weldability analysis system developed in this paper can be divided into five separate stages, as can be seen in Fig. 1. The first stage deals with the collection of all the parts in the assembly, separating them into solid body objects. In the second stage, solid bodies are matched two at the time to locate intersecting lines between them. The third step involves exporting the model to a separate binary STL file to allow for analysis based on facet geometry. For the fourth step, cross-sections are generated using arbitrary points on the intersection line, and the line direction to define the center point and normal vector of the slicing plane. In this work, the center point of the intersecting line is used. Finally, in the fifth step, light rays in a full circle given a defined resolution of spacing are constructed. Based on the number of consecutive intersections and lack-of in these lines, the spaces that are open to weld tools can be determined.

The presented method in this work can be applied to all types of geometries given a CAD environment and geometric models consisting of either large assemblies or solid bodies models. Furthermore, the weld lines can in theory from any STEP file, as the analysis is based on a

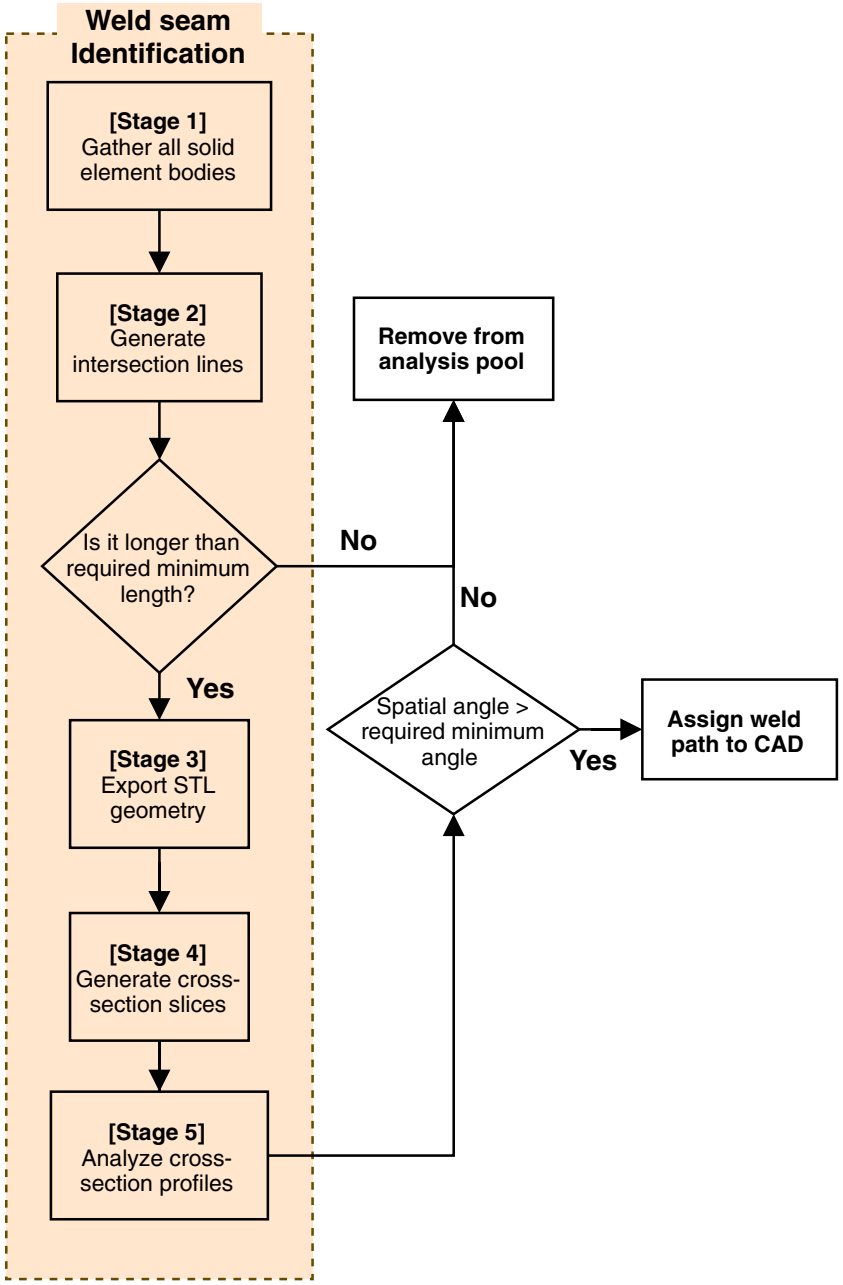


Fig. 1 Weldable curves recognition architecture

hybrid combination between CAD APIs and analysis of facet geometry. As the approach uses a standard STEP file as input, it should be applicable to any CAD software with API extensions. The underlying approach and algorithms for recognition of the weldable lines are detailed in the following subsections.

To develop the system presented in this paper, Siemens NX was used as a CAD platform along with its NXOpen Python API, which allows CAD functions to be accessed and approached in a programmatic manner. Thus, the method in stages one, two, and three will be presented with respect to functions available in the Siemens NX environment. Siemens NX allows users to define welds using their *Weld Assistant* environment. Commonly, engineers use the *Welding Joint* tool shown in Fig. 2 to define weld paths. The weld objects can be picked

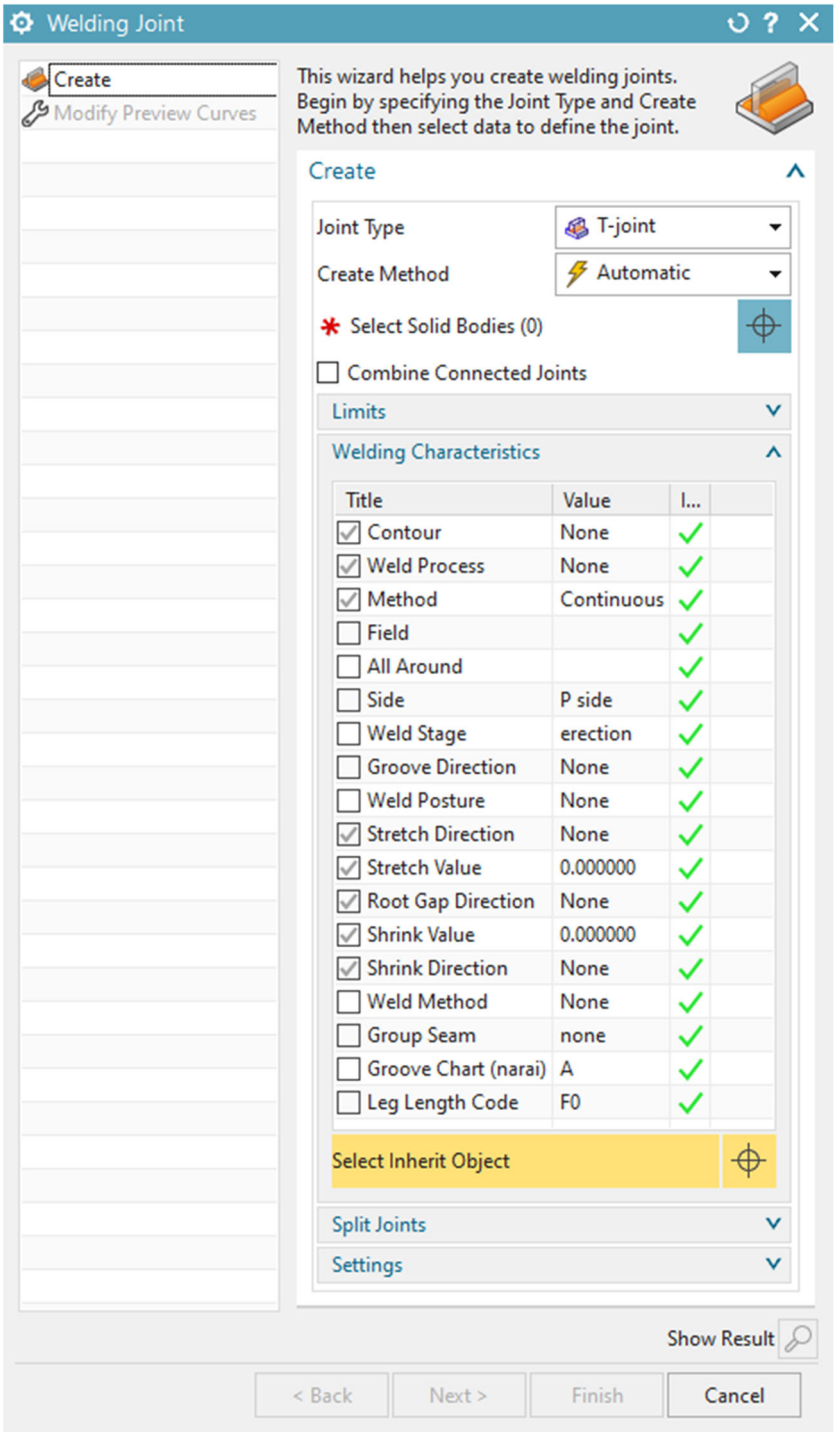


Fig. 2 Welding joint welding characteristics

up in different stages of product development with the Siemens' Teamcenter PLM system, and carry different characteristics that can be useful for other tool sets, such as the NX manufacturing environment [29]. The weld path recognition system allows for the output of these objects. Additionally, these weld characteristics can be accessed in higher detail through the API, allowing custom annotations to be assigned.

3.1 Identification of solid bodies

The identification process aims to locate all relevant solid body models for analysis in the CAD environment. Several approaches can be adopted to achieve this depending on the CAD tool. In Siemens NX, this is achieved by utilizing the NXOpen Python API.

In NXOpen, no direct function exist to collect solid body objects for assemblies. The workaround adapted in this paper utilizes the *View* class with the *AskVisibleObjects()* function to return every single object of a given scene. Consequently, the scene has to be adjusted to fit the entirety of the assembly beforehand so that all parts are included. The *Fit()* function was used to programmatically fit to view. Furthermore, as all object types are returned from the scene, solid bodies had to be filtered out. To allow usage of CAD functions in assemblies, component bodies had to be promoted. This is done programmatically with the *CreatePromotionBuilder()* function.

3.2 Generating intersection curves

The intersection curves are used as a baseline for potential weld paths. These curves can be generated by using the *Intersection Curve* tool in NX. The *IntersectionCurveBuilder* class was used as the NXOpen equivalent. As input, two solid bodies are required. Combinatorics is used to explore every combination of two bodies. Intersection curves for bodies with intentional spatial gaps can be detected by modifying the *CurveFitData.Tolerance* attribute.

Figure 3 shows intersection curves automatically generated for an arbitrary model. The yellow lines represent areas where edge geometry is overlapping with another surface. It can be observed that several detected intersection lines are suitable for welding; however, some occur inside pocket geometries and cannot be reached by welding tools. These unweldable geometries are hard to pin down in larger and complex assemblies, as other parts can become physical obstacles for the welding robot.

As intersection curves are generated as part of separate bodies, some lines naturally run underneath existing geometry. To avoid weld lines running full distances underneath other parts, the generated intersection curves had to be segmented. To achieve this, the following steps are used.

1. Loop through each intersection curve:
2. New loop of each intersection curve and calculate for intersecting lines
3. At each unique intersection, divide line into segment

3.3 Translation to facet geometry

Stereolithographic (STL) geometry is then exported using the CreateStlCreator class in NX Open. STL is often known with backronyms such as Standard Triangle Language or Standard Tessellation Language [30]. Geometrical shapes are represented using triangular facets in XYZ coordinates generated using a tessellation process. There are generally two types of STL formats, binary and ASCII. For the purpose of this work, binary was used for the better performance. The anatomy of a binary STL format can be seen in Listing 1. Binary STL formats begin with an 80-character header contrary to ASCII STL to avoid incorrect file type assumptions in software. Following the header is a 32-bit unsigned integer to indicate the number of facet in

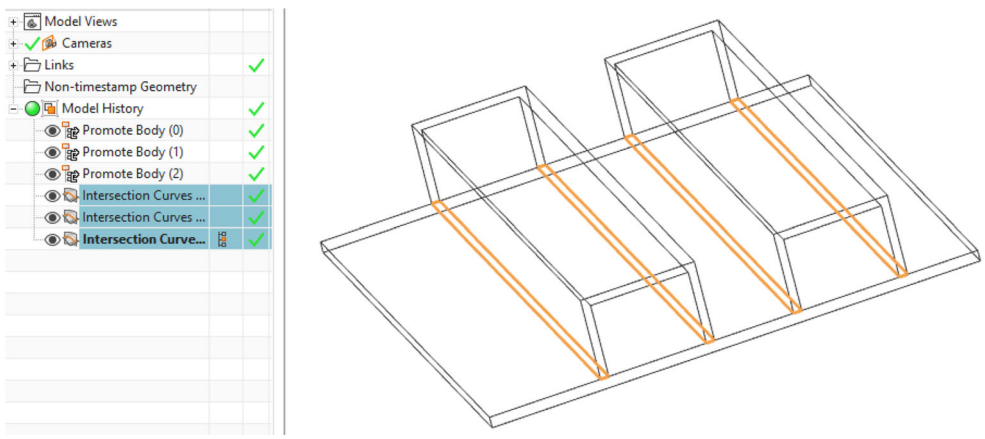


Fig. 3 Automatically generated intersection curves

```
UINT8[80] – Header
UINT32 – Number of triangles

foreach triangle
    REAL32[3] – Normal vector
    REAL32[3] – Vertex 1
    REAL32[3] – Vertex 2
    REAL32[3] – Vertex 3
    UINT16 – Attribute byte count
end
```

Listing 1 STL binary format

the file. After that is the data describing each facet. After the last triangle, the file ends. Each triangle is described by a normal vector, and three vertex positions for each corner. Each x, y, and z coordinate is defined by a 32-bit floating point number [31]. In the approach, the vertices as v_0, v_1, and v_2 are used to generate cross-sections for the welds.

Algorithm 1 Slicing of arbitrary planes.

Input: plane center point $\mathbf{p}$, plane normal vector $\vec{\mathbf{n}}$, STL binary file
Output: A list of pairs corresponding to lines in a sliced plane
for *each triangle* f_i *in triangles* **do**
  for *each vertex* V_i *in* f_i **do**
    $Q = \vec{\mathbf{n}}_x(\mathbf{p}_x - v_{i,x}) + \vec{\mathbf{n}}_y(\mathbf{p}_y - v_{i,y}) + \vec{\mathbf{n}}_z(\mathbf{p}_z - v_{i,z})$
  end
  Create list of combination of vertex V_i in triangle with length two
  for *each combination* (Q_{V_A}, Q_{V_B}) *in combinations* **do**
    if $(Q_{V_0} < 0$ *and* $Q_{V_B} > 0)$ *or* $(Q_{V_0} > 0$ *and* $Q_{V_B} < 0)$ **then**
      Interpolate point P between V_A and V_B where line intersects with slicing plane
      Append point to pairs list
    end
  end
  for *each* Q_{V_i} **do**
    if Q_{V_i} *is equal to 0* **then**
      Append vertex V_i to pairs list
    end
  end
end
return *list of pairs*

3.4 Generating cross-sections

To generate the cross-sections, slicing techniques similar to what is used to generate layers for 3D printing are used. These techniques are, however, generally developed with the slicing plane being static in the Z-plane. As the cross-sections may occur at any arbitrary direction and plane, Algorithm 1 is utilized to slice at any arbitrary planes.

The slicing plane is defined with a central point $\mathbf{p}$ along with a corresponding normal vector $\vec{\mathbf{n}}$ as depicted in Fig. 4. Using these parameters, triangle vertices can be calculated with respect to the arbitrary plane following Eq. 1. When Q is equal to zero, the arbitrary vertex point is coplanar to the defined plane. This can be used to check whether two points in space correspond to opposite sides of a given plane. If two values of Q are calculated to opposing positive or negative signs, the points are on opposite sides.

$$Q_i = \vec{\mathbf{n}}_x(\mathbf{p}_x - v_{i,x}) + \vec{\mathbf{n}}_y(\mathbf{p}_y - v_{i,y}) + \vec{\mathbf{n}}_z(\mathbf{p}_z - v_{i,z}) \quad (1)$$

If two vertices of a triangle occur on different sides of the plane, interpolate to the coplanar point of the line formed between the two. Similarly, when Q is equal to zero, the vertex position can be used directly as a slicing point. As input for Algorithm 1, the center point in the intersection curve is defined as $\mathbf{p}$ and intersection curve direction as $\vec{\mathbf{n}}$. Following this procedure for every triangle, we are able to extract any cross-section defined by any arbitrary plane. Finally, we transform the sliced points into XY coordinates using a transformation matrix to allow ray scanning algorithms to function more effectively. In Fig. 5d and c we see the resulting unique outputs from this method when applied to two the corresponding assemblies (Fig. 5a and b).

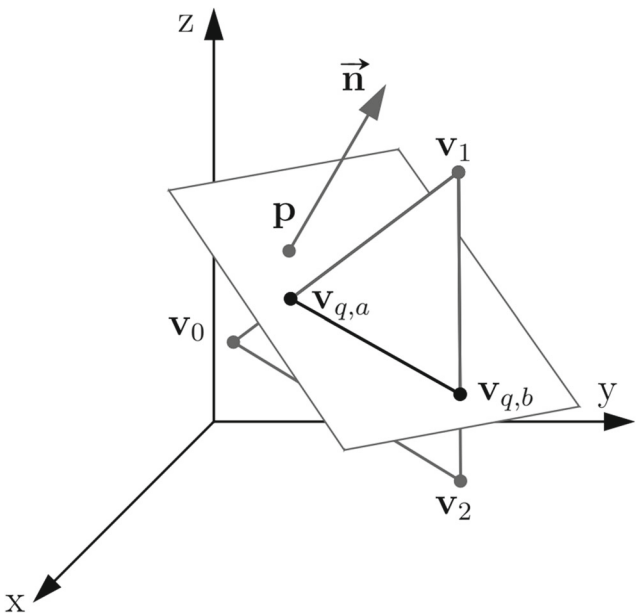


Fig. 4 Arbitrary slicing plane

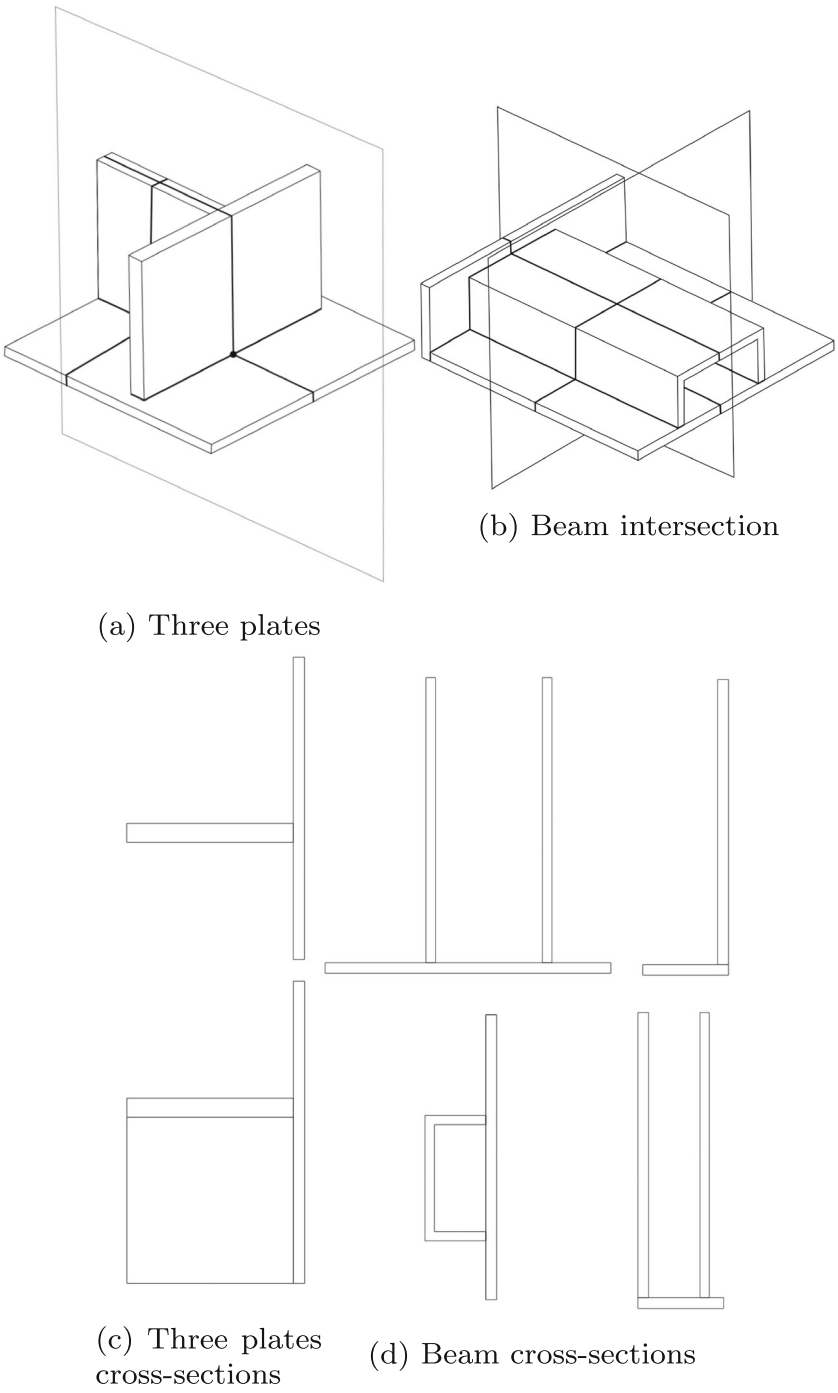


Fig. 5 Cross-sections generated from intersection curves on test components. **a** Three plates. **b** Beam intersection. **c** Three plates cross-section. **d** Beam cross-sections

3.5 Cross-section analysis and path filtration

This inspection process aims to determine whether a weld is feasible or not based on the cross-section geometry. This is mainly done by assessing the open space surrounding the weld point in 2D. First, we define the weldability of a given path by the amount of consecutive open space around the weld. The openness is defined by a line by given length extending from a center without intersecting with cross-section geometry. Furthermore, this line has to exist on the outside of a solid body. A point in polygon (PIP) algorithm known as the crossing number algorithm is used to determine this. The step details are outlined below.

1. Generate two list of points of resolution d at radius r_t and r_m
2. Loop through both lists simultaneously, in which:
   (a) Loop though cross-section lines, in which:

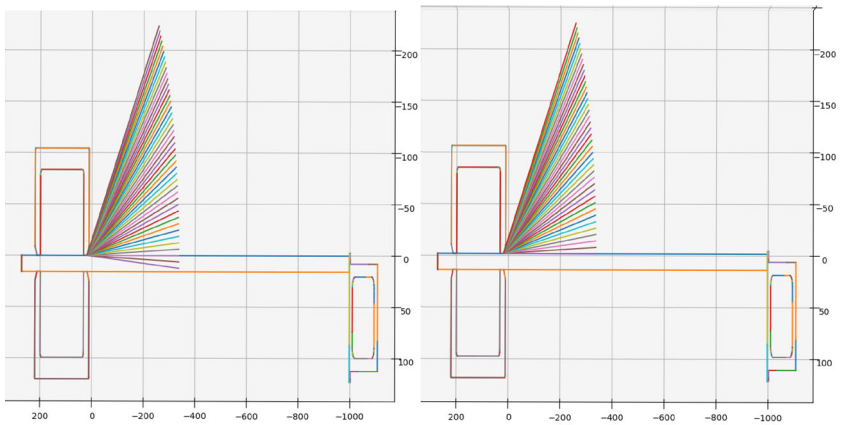
i Check line between point r_{t_i} and r_{m_i} for intersections;
ii If intersection exist, break loop and set intersection to True;

(b) If line has no intersection, check if point r_{m_i} is a point in polygon;
(c) If line has no intersection and previous line has, append new counting instance to list of open spaces;
(d) If line and previous line has no intersection, and point r_{m_i} is not in a polygon, increment ray scan to open space;

3. Scan list for largest number l_i;
4. Return open spatial angles in degrees $\phi_o = 360 * l_i/d$;

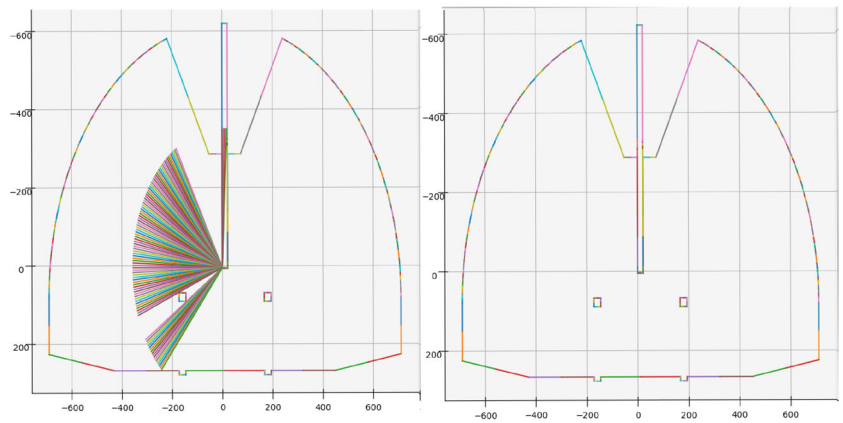
Following the described steps, we are able to retrieve an approximate maximum angle ϕ_o of open space for every cross-section. By increasing the input parameter d the potential accuracy for ϕ_o increases. Finally, we compare ϕ_o to a selected minimal spatial angle value to filtrate out unqualified weld paths. To determine whether a line is within a closed polygon (2D), or inside solid body (3D), we generate a line from the endpoint r_m with a length of practical infinity. Then, every line in the 2D slice is looped to check for intersections. If the number of intersections found is an even number, the point is assumed to be outside a closed polygon.

Figure 6 depicts the valid rays found by the algorithm when used on example components. In Fig. 6a and b, the practical implications of the PIP check can be observed. In some cases, depending on the resolution selected for d, lines. Furthermore, as seen in Fig. 6c and d, large and flat slices are specially susceptible to yielding incorrect results without PIP. Variables r_t and r_m determine the distance from the center where a ray starts and ends at which line intersections are checked respectively.



(a) Example assembly without PIP check

(b) Example assembly with PIP check



(c) Gusset node without PIP check

(d) Gusset node with PIP check

Fig. 6 Ray casting plotted on example components. **a** Example assembly without PIP check. **b** Example assembly with PIP check. **c** Gusset node without PIP. **d** Gusset node with PIP check

4 Implementation and result

The method presented in this paper was implemented in a Siemens NX CAD environment by running journals written in the NXOpen Python API. The automatic recognition of potential weld paths was then tested in a number of different assembly geometries, two of which were real use cases. The models were loaded in from STEP files as pure assembly geometries. As parameters for the method in this demonstration the minimal required spatial openness angle set to 40°, and minimum weld path to 40 mm. The ray scan length r_m was set to 300 mm, and tolerance r_t to 5 mm.

As an initial trial, some commonly used types of weld joints were first tested the system. Figure 9 depicts welding paths recognized on these kinds of joint geometries.

To benchmark the approach, a comparison was made to the *Welding Joint* with automatic path inference available in NX. The models used for the benchmark can be seen in Fig. 7. The aforementioned parameters were used for the approach. For the Welding Joint builder, the *Create Method* was set to automatic and all the bodies were selected as input for each assembly shown. For the comparison, the number of segmented weld paths found by each method were then extracted. As the solution baseline, weld paths were

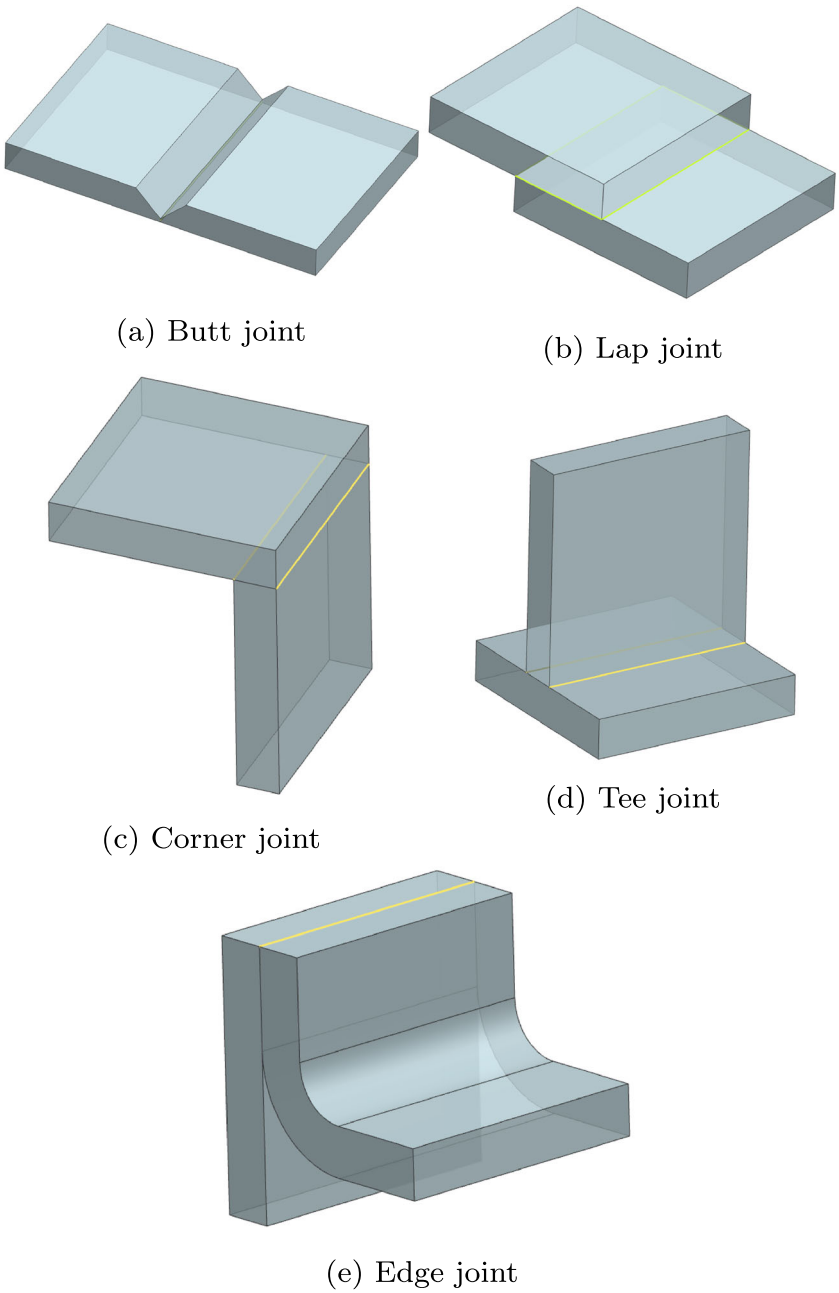


Fig. 7 Demonstrative assemblies. **a** Butt joint. **b** Lap joint. **c** Corner joint. **d** Tee joint. **e** Edge joint

Table 1 Our approach vs built-in solution benchmark

Model	Parts	Built-in solution	Our method	Manual (solution)
Three plates	3	3	7	7
Beam intersection	3	4	8	8
Stiffener plate	4	6 (6 unreachable)	14	14
Gusset node	6	16	40	40
Fillets and chamfers	4	6 (4 unreachable)	16 (2 unreachable segments)	18
Short beam	7	13	29	30
Stiffening section	19	34	68	68
Stiffened wall	11	32 (14 unreachable)	68	68

manually selected. The number of weld paths identified for each method can be seen tabulated in Table 1. Finally, the paths generated by the approach are shown in Fig. 8 as yellow lines. The use case models can be seen in Figs. 8c, d, and 9c, d.

5 Discussion

In this section, the viability of our approach based on the findings from Section 4 is discussed. We then outline some potential problems and weaknesses that were identified with the approach. Finally, we discuss some of the potential directions towards more intelligent systems.

5.1 Analyzing results and viability

The proposed approach was developed in a Siemens NX environment using the NXOpen Python API. Using the *Run Journal* function in NX, the code could be executed on any assembly to automatically locate and assign weld paths. The approach is purely topology-driven and does not need to rely on assistance or big changes in the workflow. As a baseline, common weld joints seen in Fig. 9 were tested first. All paths were correctly identified for these; however, different commonly used variants exist. Some of these contain deliberate spaced gaps, which can be hard to identify correctly by the intersection generation, as elaborated in Section 5.2. The assemblies depicted in

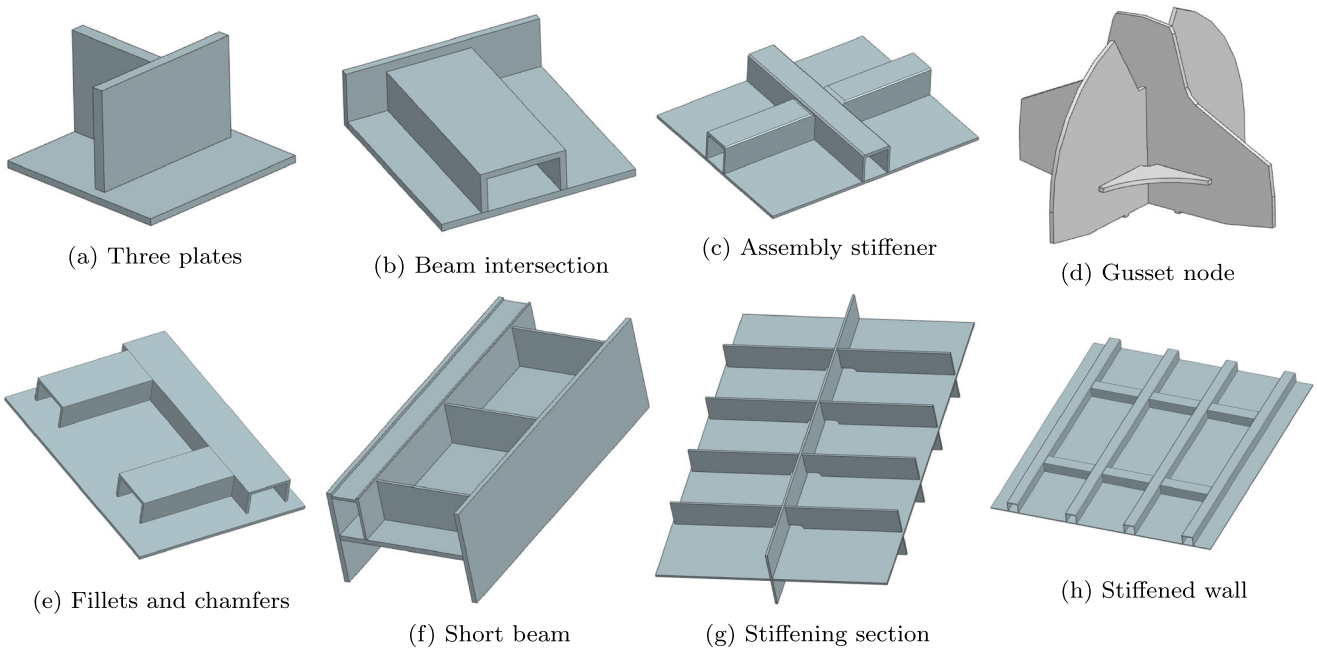


Fig. 8 Demonstrative assemblies. **a** Three plates. **b** Beam intersection. **c** Assembly stiffener. **d** Gusset node. **e** Fillets and chamfers. **f** Short beam. **g** Stiffening section. **h** Stiffened wall

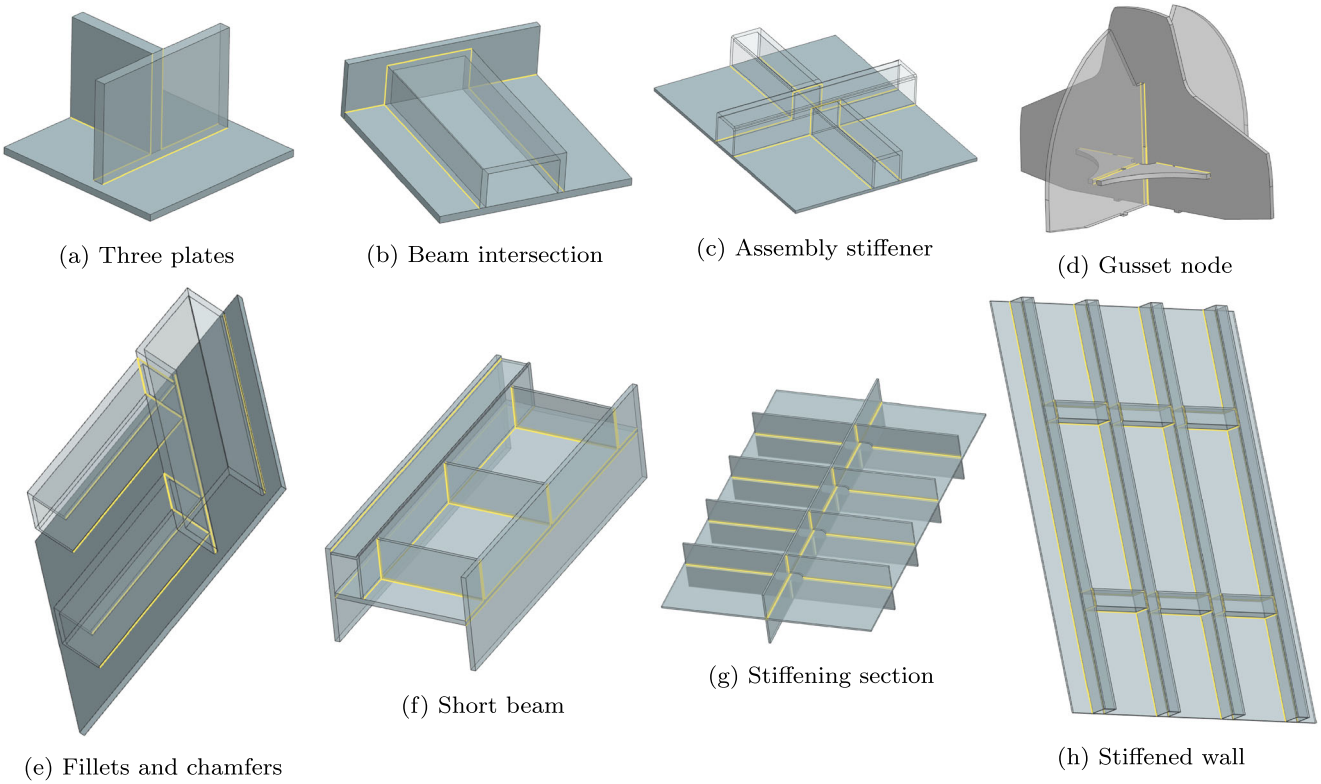


Fig. 9 Demonstrative assemblies with generated weld paths. **a** Three plates. **b** Beam intersection. **c** Assembly stiffener. **d** Gusset node. **e** Fillets and chamfers. **f** Short beam. **g** Stiffening section. **h** Stiffened wall

Fig. 7 were mainly made for the purpose of the paper, with the exception of the two aforementioned, to allow focus and clarity on relevant details and points. However, more complex geometries taken from practical cases were also tested. Of these, the largest assemblies had over 80 parts, which resulted in over 200 weld paths.

With feature recognition systems, defining a clear level of accuracy is difficult, and something the authors struggled with in a practical sense. By proposing a generalized weld-recognition tool, the allowed input is essentially a free design space which almost guarantees unpredicted outcomes. This is a common problem for feature recognition systems [32]. As such, we instead compared our solution against the weld path inference tool available in Siemens NX using weld paths defined manually by a human as an optimal score. The scores are tabulated in Table 1. When compared to the automatically inferred weld paths generated by the built-in solution, Welding Joint builder, our method is shown to be capable of recognizing and applying more weld paths for all the cases presented. This can be seen in Table 1. Similarly to the Welding Joint builder, we are able to generate the same type of weld objects, allowing the system to be more easily adapted into existing PLM systems. In addition to finding almost all the weld paths, no lines were incorrectly identified as opposed to the welding joint builder tool.

To our knowledge, not many weld path recognition systems have been developed to function within existing CAD environments. This sentiment is shared by another work published by Kiani and Saeed in 2019 [27], dealing with automatic recognition of spot weld locations in CAD. Similarly, APIs were used to integrate their tool with the intention to bridge the gap between CAD and CAM. Likewise, the authors believe that there are advantages and potentials in such systems despite the current lack in published work. Of the few found with some relevancy, most focus on the characterization, such as [15] and [10]. Furthermore, publicly available industrial implementations and commercially available feature recognition systems exist [33]. This work has focused on adapting feature recognition techniques from the works outlined in Section 2 such as [1, 18, 28] into welding applications in the CAD environment. This has led to restrictions in comparing the method to existing approaches, with the exception of the welding joint tool in NX. Furthermore, the welding joint tool seems to be tailored for ship-related structures, which might lead to an unfair comparison [29]. Even so, based on the benchmarking seen in Table 1, our approach seems

capable of finding more potential weld paths than the built-in solution, even for cases similar to ship structures.

5.2 Known weaknesses and potential solutions

The models used to demonstrate the system range from very simple cases to more complex. Still, it is not unreasonable to assume that some very complex assemblies can become challenging for the system, as the assumption used to detect welds is simple. In 2D, we are not able to determine whether a weld gun can reach certain areas from the side, which might be used in some designs. Furthermore, cross-sections only are generated from single points on the intersection curves, it is possible that some unfortunate positions are analyzed. This is observed in Fig. 8h. In our demonstration, the point used for cross-section generation was set to be at the center of the curve. In topologies with perfect symmetry, this might cause a problem if blocking objects happen to be placed in the middle. However, this can be avoided by offsetting the slicing point, or adding more slicing points to each intersection curve at the cost of more processing time. Currently, all parts of the assembly are considered simultaneously for the analysis. In some constructions, the assembly order allows certain welds to be accessible before everything has been assembled. This can be dealt with by only considering two intersecting parts at the time. However, more complicated systems might be needed to assess the assembly order as well.

Figure 10 is a side-view depiction of the model shown in Fig. 8e, containing chamfer and fillet geometry. Based on the findings shown in Table 1, two weld paths are missing. When looking back to the side-view, it can be observed the fillet geometry keeps the intersection curve from occurring at a joint location between the three bodies. This in turn leads to the segmentation step to fail detecting the particular junction so that paths can be divided properly. When this happens, the weld path is can be seen continuing as a single path throughout the geometry as shown in Fig. 9e. To solve this, several approaches can be adapted. One solution is to extend the lengths of the intersection curves with a given tolerance such as the maximum edge blend radius. However, this has yet to be tested.

Fig. 10 Side-view of fillet geometry shown in Fig. 8e leading to no intersection for segmentation

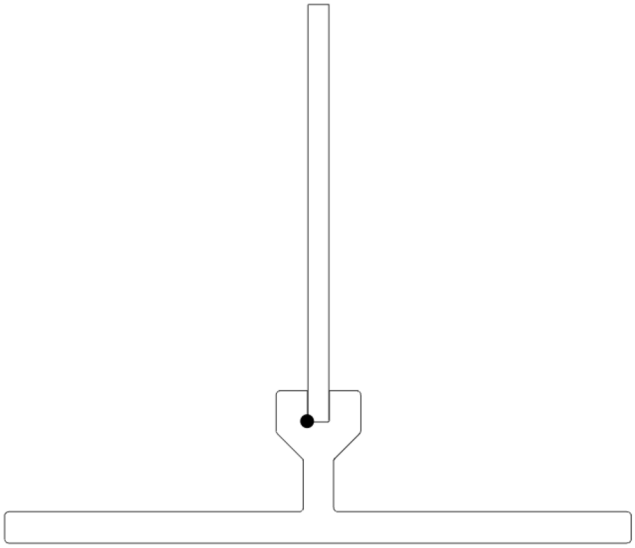


Fig. 11 Cross-section view of a joint type leading to undesired intersection positions inside the joint geometry (dot represents current intersection curve)

Because the analyzed point is based on the intersection curves, some types of joints are difficult identify to with the method. The intersection curve in Fig. 11 is one such example. The intersection curve, represented with a black dot, occurs deep inside the joint for this geometry. The result from the ray-casting approach results in an open spatial angle for zero.

In some designs, certain locations are only accessible before the assembly of other parts. One such design can be seen the short beam example depicted in Fig. 8f. These parts rely on specific sequences where previously accessible weld paths become closed off by other parts welded on later. The same reason explains why one weld path is "missing" in Table 1 for the short beam example. As seen in Fig. 9g, 29 out of 30 weld paths were found. The "missing" weld path is located inside the same path represented by a black dot in Fig. 12. That is because an entire structure is considered, so that ray casting would naturally find an obstacle - the other solid body of the assembly. One way to overcome this is to generate cross-sections using only the two bodies relevant to the intersection curve knowing that we deal with an assembled structure.

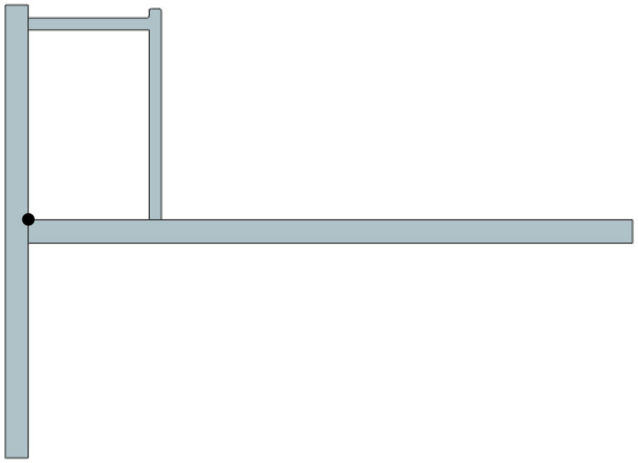


Fig. 12 Cross-section view of part shown in Fig. 8f (simplified) showing an intersection that can be welded given a correct sequence of assembly (dot represents current intersection curve)

5.3 Expanding the system and future directions

The focus has been aimed towards developing a recognition system for weld paths. However, for such a system to be truly useful, the weld paths should be characterized, documented, and further analyzed. The approach developed is intended to function as a potential cornerstone for larger inference systems down the line. Systems such as these can build upon the weld path input generated by the approach using knowledge, and rule-based logic. The problem with many feature recognition techniques is the unpredictability of allowing a free design space and geometries, which generally require assumptions, unless a problem is fully modelled and simulated [32]. In future iterations, the authors plan on incorporating a knowledge-base into the framework to allow cross-sections to be referenced. This could allow for specific weld types belonging to particular cross-sections to be recognized and thus characterized. From there, tools and functionality as envisioned in [3] can be realized.

6 Conclusion

This study proposes an approach to automatically detecting weld paths in assemblies within a CAD environment. The approach developed using a commercially available CAD environment, Siemens NX, along with the NXOpen Python API has been presented. Finally, demonstrations on differing geometries have been provided along with a benchmark against a native welding joint auto-inference tool.

The research was conducted on a commercially available tool, Siemens NX. This may offer limitation when attempting to implement similar methods on other platforms, as technical API may vary in capabilities. However, the generalized approach should be applicable.

In all the examples provided, the method performs near perfectly. The only exceptions being that as it only considers what is weldable from a 2D point of view. Welds that can be performed at an angle through an extended weld gun cannot be detected. Furthermore, as the weld path recognition is purely based on topological analysis as it currently stands, all lines that pass the filtration are automatically assigned a weld path. In some cases, this might not be wanted. These obstacles can, however, be dealt with by adding additional methods proposed in the discussion. Furthermore, future iterations will combine knowledge-based inference with the topology analyzation method proposed in this paper to improve the accuracy and knowledge capturing capabilities.

Funding Open access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital). The research is supported with the project (number 295138) funded by the Research Council of Norway.

Availability of data and materials The approach and results presented are developed on NXOpen API and supported to the knowledge of the authors with relevant sources.

Code availability The code is not available as it is also a part of the project deliverables. However, algorithm is defined in the article, which was used for implementation in software.

Declarations

Conflict of interest The authors declare competing interests.

References

1. Ranjan R, Kumar N, Pandey RK, Tiwari MK (2005) Automatic recognition of machining features from a solid model using the 2d feature pattern. Int J Adv Manuf Technol 26(7-8):861–869
2. Sheen Bor-Tyng, You Chun-Fong (2006) Machining feature recognition and tool-path generation for 3-axis cnc milling. Comput Aided Des 38(6):553–562
3. Prescott SAO, Tran TA, Lobov A (2020) Automatic weld path definition in cad. Procedia Manuf 51:478–484

4. Autodesk (Org.). Autodesk forge. https://forge.autodesk.com. Accessed: April 2. 2021
5. Solidworks (Org.). Solidworks API Support. https://www.solidworks.com/sw/support/api-support.htm. Accessed: April 2. 2021
6. Lobov A, Tran TA (2020) Object-oriented approach to product design using extended nx open api, vol 51, pp 1014–1020. 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021)
7. Hayasi MT, Asiabanpour B (2009) Extraction of manufacturing information from design-by-feature solid model through feature recognition. Int J Adv Manuf Technol 44(11-12):1191–1203
8. Lavrentyeva MV, Chimitov PY (2017) Implementation of recognition algorithm with nxopen api in siemens nx. In: 2017 international conference on industrial engineering, applications and manufacturing (ICIEAM). pp 1–4
9. Sommerville MGL, Clark DouglasER, Corney JR (2001) Viewer-centered geometric feature recognition. J Intell Manuf 12(4):359–375
10. Kuss A, Dietz T, Ksensow K, Verl A (2017) Manufacturing task description for robotic welding and automatic feature recognition on product cad models. Procedia Cirp 60(1):122–127
11. Garcia F, Lanz M, Järvenpää E, Tuokko R (2011) Process planning based on feature recognition method. In: 2011 IEEE international symposium on assembly and manufacturing (ISAM). IEEE, pp 1–5
12. Hillbrand C, Frank G (2012) Knowledge-based automated programming of welding robots for lot-size one products. In: Engineering systems design and analysis, vol 44878. American Society of Mechanical Engineers, pp 27–36
13. Bi ZM, Lang SYT (2007) A framework for cad- and sensor-based robotic coating automation. IEEE Trans Ind Inf 3(1):84–91
14. Tikhomirov VA (2019) The method of automatic determination of the types of spatial angles in 3d models of cad systems. In: 2019 international science and technology conference "EastConf", pp 1–4
15. Pabolu V, Stolt R, Johansson J (2016) Manufacturability analysis for welding: A case study using howtomation© suite. In: Proceedings of the 23rd ISPE Inc. international conference on transdisciplinary engineering, Parana, Curitiba, October 3–7, 2016
16. Shi P, Qi Q, Qin Y, Scott PJ, Jiang X (2020) A novel learning-based feature recognition method using multiple sectional view representation. J Intell Manuf :1–19
17. Hansen LK, Salamon P (1990) Neural network ensembles. IEEE Trans Pattern Anal Mach Intell 12(10):993–1001
18. Frank MC, Wysk RA, Joshi S (2005) Determining setup orientations from the visibility of slice geometry for rapid computer numerically controlled machining. J Manuf Sci Eng 128(1):228–238
19. Shah JJ, Anderson D, Kim YS, Joshi S (2000) A discourse on geometric feature recognition from CAD models. J Comput Inf Sci Eng 1(1):41–51
20. Li Y (2008) Manufacturability analysis for non-feature-based objects. Digital Repository@ Iowa State University, http://lib.dr.iastate.edu/
21. Gao S, Zhao W, Lin H, Yang F, Chen X (2010) Feature suppression based cad mesh model simplification. Comput Aid Des 42(12):1178–1188
22. Feng C, Liang J, Ren M, Qiao G, Lu W, Liu S (2020) A fast hole-filling method for triangular mesh in additive repair. Appl Sci 10(3):969
23. Wang Y, Liu R, Li F, Endo S, Baba T, Uehara Y (2012) An effective hole detection method for 3d models. In: 2012 Proceedings of the 20th European signal processing conference (EUSIPCO), pp 1940–1944
24. Yao Y, Kong L, Hu Q (2009) A boundary identification method for 3d closed triangle mesh. In: 2009 IEEE 10th International conference on computer-aided industrial design conceptual design, pp 870–873
25. Noble JA, Gupta R, Mundy J, Schmitz A, Hartley RI (1998) High precision x-ray stereo for automated 3d cad-based inspection. IEEE Trans Robot Autom 14(2):292–302
26. Hase VJ, Bhalerao YJ, Verma S, Jadhav S, Vikhe Patil GJ (2018) Complex hole recognition from cad mesh models. Int J Manag Technol Eng 8(9):1102–1119
27. Kiani MA, Saeed HA (2019) Automatic spot welding feature recognition from step data. In: 2019 international symposium on recent advances in electrical engineering (RAEE), vol 4. IEEE, pp 1–6
28. Li Y, Frank MC (2012) Computing axes of rotation for setup planning using visibility of polyhedral computer-aided design models, vol 134
29. Siemens PLM (2014) Software Solid Edge
30. Grimm T (2004) User's guide to rapid prototyping. Soc Manuf Eng
31. Iancu C, Iancu D, Stăncioiu A (2010) From cad model to 3d print via" stl" file format Fiability & Durability/Fiabilitate si Durabilitate
32. Shi Y, Zhang Y, Xia K, Harik R (2020) A critical review of feature recognition techniques. Comput-Aid Des Appl 17(5):861–899
33. Babic B, Nesic N, Miljković Z (2008) A review of automated feature recognition with rule-based pattern recognition. Comput Ind 04(4):321–337