

Sarah Ann Oxman Prescott

Design of a KBE system for automatic weld path definition in CAD

Master's thesis in Mechanical Engineering

Supervisor: Andrei Lobov

January 2020

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering



Norwegian University of
Science and Technology

Sarah Ann Oxman Prescott

Design of a KBE system for automatic weld path definition in CAD

Master's thesis in Mechanical Engineering
Supervisor: Andrei Lobov
January 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering



Preface

This MSc thesis concludes a five-year degree in Mechanical Engineering through the integrated Masters program at the Norwegian University of Science and Technology (NTNU). The work was conducted at NTNU in Trondheim in the fall of 2019 as part of a research project funded by the Norwegian Research Council. Collaborators have included Hydro Aluminium AS, Leirvik AS, Marine Aluminium AS, DIGITREAD AS and Fjellstrand. The project is still in its early stages, thus far collaboration has been mainly with Leirvik and Marine Alumium. Both groups have provided CAD-models for analysis and formalized industry needs in relation to robotic welding.

This report requires a basic understanding of CAD, programming and manufacturing.

20 January 2020

Acknowledgments

Professor Terje Rølvåg introduced me to the research project which now includes this thesis. He has sparked enthusiasm amongst all of the project's participants. Pre-masters students Morten Fossmark and Mathias Risstad have been excellent sparring partners and supportive on a daily basis. PhD-students Tuan Anh Tran, Tord Hansen Kaasa and Jaime Marco Rider who are involved in various aspects of the project and who contributed to a complete solution objective, have been available for consultation. Professor Andrei Lobov has led the team through clouds of uncertainty and ambiguity, motivating us to take action and encouraging us to challenge boundaries. My sincerest thanks goes to all of these individuals.

I am also grateful to Ole Terje Midling and all the engineers that welcomed us at Marine Aluminium. Thank you to both Marine Aluminum and Leirvik for supplying us with CAD-models to test our solutions and for information on industry challenges and needs.

S.P.

Abstract

Knowledge based engineering (KBE) is often described as the capture and re-use of engineering knowledge. One instrumentalisation of the concept is digital storage and application of engineering knowledge in code form. Coding skills are not required for use of computer-aided design (CAD), which may be perceived as an advantage. It may seem counterintuitive to work with CAD programmatically rather than interactively. However, programmatic use of CAD offers major benefits related to KBE. Programming allows for dynamic definitions of solutions; restrictions can be defined as conditional statements. Despite KBE's potential benefits and success in the automotive and aerospace industries, its use is not an industry norm. Many attribute this to the cost in time of applying KBE practices. KBE has been under development since the 1980s. Several cases of KBE have been presented in literature. These often endorse KBE boasting of its potential, but neglecting to address the specific challenges associated with its implementation. This work uses CAD programmatically to enable automation and KBE for the case of large welded aluminium structures. The design of a program is presented: a Python script that can be run on a CAD assembly in NX Siemens to identify potential weld lines and export a weld path based on these for automatic generation of robot code for a welding robot.

Sammendrag

Knowledge-based engineering (KBE) er ofte beskrevet som lagring og gjenbruk av ingeniørfaglig kunnskap. En instrumentalisering av begrepet innebærer digital lagring og anvendelse av ingeniørkunnskap i form av kode. Kodeferdigheter er ikke nødvendig for bruk av computer-aided design (CAD), noe som kan sees på som en fordel. Det kan virke kontraintuitivt å bruke det visuelle verktøyet via koding heller enn interaktivt. Men bruk av CAD i kombinasjon med kode byr på fordeler i implementering av KBE. Programmering tillater dynamiske definisjoner av løsninger; spesifikasjoner kan defineres som betingelses-baserte linjer med kode. På tross av potensielle fordeler med KBE og suksess i bil- og romferdsindustri, er bruken ikke standard i produktutvikling i industrien per i dag. Mange mener dette er fordi det er uforholdsmessig tidkrevende å innføre KBE. KBE har vært under utvikling siden 1980-tallet. Flere eksempler har blitt presentert i litteraturen. Ofte blir det skrytt av potensialet til KBE, men uten forklaring på hvorfor det er lite brukt. Dette arbeidet bruker CAD i kombinasjon med kode for å tillate automatisering og KBE anvendt på robotsveisning av store aluminium strukturer. Design av et program er presentert: et script i Python, som kan kjøres på STEP-filer i NX Siemens for automatisk identifisering av potensielle sveiselinjer og som kan eksportere disse linjene for senere bruk til automatisk generering av robotkode.

Contents

Preface	i
Acknowledgments	ii
Abstract	iii
Sammendrag	iv
Contents	v
List of Figures	vii
Code Listings	viii
Abbreviations	ix
1 Introduction	1
1.1 Knowledge-building project for Industry – MAROFF	2
1.2 Scope	2
1.3 Research questions	3
1.4 Structure of thesis	3
2 State of the art	6
2.1 Programmatic CAD & CAD-APIs	8
2.1.1 Knowledge-based engineering and automation	10
2.1.2 Platform independence and extendability	15
2.1.3 Industrie 4.0	17
2.2 Engineer-to-order and small-medium enterprises	19
2.2.1 Current welding practices and challenges	20
2.2.2 Robotic welding	21
2.3 Summary and thesis contribution to state of the art	24
3 Methodology	25
3.1 System architecture	25
3.2 Flowchart of system processes	26
4 Implementation	28
4.1 Main tasks to be automated	30
4.1.1 Find potential weld locations	30
4.1.2 Mark and store weld path	30
4.1.3 Export weld path	31
4.2 Code	32
4.2.1 Software- and language-specific system flowchart	32
4.2.2 Pseudocode	34
4.2.3 Coding subtasks	35

5 Results	36
5.1 Subtask results	36
5.2 Proposal for tests of completed system	39
6 Discussion	40
6.1 Interpretations and implications of findings	40
6.2 Answering research questions	40
6.3 Strengths and weaknesses	41
7 Conclusion	44
7.1 Future work	44
Bibliography	45
A Code and testing	49
A.1 Program listings for subtasks	49
B Publishability	58
B.1 MTP-conference poster	58
B.2 FAIM 2020 (1) abstract accepted, paper to be submitted by February 2020	58
B.3 FAIM 2020 (2) paper submitted	58

List of Figures

1	Thesis scope	5
2	Structure of state of the art chapter	7
3	RAMI [1]	18
4	RAMI vertical layers [1]	18
5	Hagen’s solution flowchart [2]	23
6	System architecture	25
7	Flowchart showing main modules	27
8	Structure of implementation chapter	29
9	Software- and language-specific system flowchart	33
10	Subtask c. Loop through model components (demonstrated on bridge element)	37
11	Subtask c. Loop through model components (demonstrated on living quarters corner)	37
12	Subtask d. Locate intersection lines between components	38
13	Subtask h. If-statement in loop (demonstrated on engine blower)	38
14	User interface design	43
15	MTP-conference poster 2019	59
16	Accepted abstract for FAIM 2020	60
17	<i>Formalization of engineering knowledge for industrial robots using Knowledge Fusion language</i>	61

Code Listings

A.1	Subtask a. Interact with STEP-file	49
A.2	Subtask b. Separate elements in STEP-file assembly	49
A.3	Subtask c. Loop through model components	50
A.4	Subtask d. Locate intersection lines between components	50
A.5	Subtask h. If-statement in loop	57

Abbreviations

Abbreviations

API Application programming interface

CAD Computer-aided design

ETO Engineer-to-order

I4.0 Industrie 4.0

KBE Knowledge-based engineering

KBP Knowledge-building project for industry

KF Knowledge Fusion

M.A. Marine Aluminium

MOKA Methodology and software tools oriented to knowledge-based engineering applications

OLP Offline programming

RAMI 4.0 ReferenceArchitecture Model Industry 4.0

STEP Standard for the Exchange of Product Data

1 Introduction

Product and process knowledge is often stored in various formats at different stages of product development. There is not always a predefined way of expressing manufacturing constraints in a design model and vice-versa. Expressing information from one software program in another is not always straightforward. This has consequences for the development of an automatic system. The format in which knowledge is expressed – orally, in plain English, in code – also has consequences for storage and re-use, the aim of knowledge-based engineering (KBE).

KBE can save time by, for example, storing engineering knowledge in code for later use. Automation can save time by partially or fully removing human interaction from a process. Code can be used for both KBE and automation. Computer-aided design software (CAD) is used to generate virtual three-dimensional models of a product in a visual "click-and-drag"-type environment where coding skills are not required. There are, however, options for adding programming logic in design of, and interaction with, a CAD model, which will be a theme throughout this thesis. Code is used to both add knowledge to, and extract knowledge from, CAD software. In this work, a program has been designed to add welds to a CAD assembly model and extract the weld path to be processed in robot programming software for automatic robot code development.

The language or format in which engineering knowledge is available, is not only significant for transfer of knowledge between software and functions in product development; it also matters for how knowledge can be processed and operated on. Geometric knowledge of a product is typically represented with a CAD model. The very same geometric knowledge may be expressed in code, e.g. Python, and operated on using Python functions to process information. Knowledge available in code form is more readily integrated with third party software. In other words, code can aid in integrating a variety of software and hardware to create intelligent production facilities, as discussed in the Industrie 4.0 section of the theory chapter.

Engineer-to-order (ETO) products are by definition custom to some degree. A design or product may not be available pre-order, because the customer's requests and specifications are yet to be communicated. But what if the design of a product could be generated automatically upon receiving a customer's request? And, what if a presentation of manufacturability, adherence to standards, bill of materials, costs and timelines could be generated automatically as well? ETO companies would be able to evaluate requests and present offers in a time efficient manner. Creating the software to allow for this type of automation is a large endeavor requiring the achievement of many partial goals. This thesis takes on one of these partial goals as illustrated in Figure 1.

This work is part of a larger project that aims to automate the design and production of large welded aluminum structures: the project and its partners are introduced in the next section. The focus here is not on the specific product, but on the system designed as a step towards automation of

design and production. The system is designed to define and export a weld path with the intention of generating code for a welding robot. NX Siemens and Python are used for defining the weld path based on an assembly model imported as a STEP-file. Visual Components is intended to be used for robot code generation by collaborators.

1.1 Knowledge-building project for Industry – MAROFF

This work is part of a larger project that aims to automate the design and production of a large welded aluminum structure. Initially this structure was defined as a ship hull. However, the project has evolved and is no longer limited to a specific structure. The emphasis has shifted to advancing automation of robotic welding in general. Further delineation of the scope of this thesis is laid out in the next section. The Knowledge-project's official start was February 2019, and it is to end in July 2024. This work is part of the explorative phases and contributes to the ground work for the four-year project. An aim is to evaluate the best tools for the upcoming tasks, as well as to explore whether some tools are missing in the market.

The project is a Knowledge-building project (KBP) for Industry funded by The Research Council of Norway. It is also part of MAROFF – Innovation Programme for Maritime Activities and Offshore Operations, which, in addition to other objectives, aims to "[...]improve interactivity and knowledge transfer between the R&D community and the maritime industry" [3]. The Norwegian University of Science and Technology (NTNU) is the project owner. Industry partners listed in the project documentation are Hydro Aluminium AS, Leirvik AS, Marine Aluminium AS, DIGITREAD AS and Fjellstrand. The industry partners have specified their needs and supplied the project with CAD-models for testing and analysis.

Two pre-Masters students, Mathias Risstand and Morten Fossmark, and three PhD students, Jaime Marco Rider, Tord Hansen Kaasa and Tuan Anh Tran, have been working in parallel on different aspects of the project. Some of their work will be referred to. There has been tight collaboration and follow-up within the group. Prior to this work, a pre-Masters and a Masters thesis were written under the project by Elias Bragstad Hagen [2]. Any reference to "Hagen" in this work refers to him and his solution for automatically generating robot code. His effort laid the foundation on which this work is built. He created a system that employed automatically generated robot code for a weld trajectory defined by sketches in a CAD model. The sketches that he used were added manually to a CAD model. The present work designed a method to *automatically* identify, place, and export welds in a CAD model.

1.2 Scope

Figure 1 highlights the focus of this thesis and illustrates how it fits in with the Knowledge-building project as a whole. The scope of this thesis is limited to transfer of information to and from a CAD model at the design stage. This includes the transfer of information to and from a customer upstream and to manufacturing downstream. It also includes the capture and transfer of engineering knowledge, putting knowledge based engineering (KBE) into practice. It demonstrates one way of automating information processing and transfer between software programs. Application Program-

ming Interfaces (APIs) are used to link between modelling and robot programming software. This will be elaborated on in the state of the art and implementation chapters.

Many aspects of robotic welding, and challenges associated with aluminum in particular, are not addressed, but are mentioned in the concluding chapter 7 on further work because this work has direct and indirect consequences for other aspects of the project. As illustrated in Figure 1, other students are currently working on areas tangential to this thesis: Morten Fossmark on robot code, Mathias Risstad on analysis and Jaime Marco Rider on feedback from sensors. This work focuses on capture and application of engineering knowledge in software, putting KBE into practice and making progress towards the industry of the future, Industrie 4.0. KBE and Industrie 4.0 are described in the state of the art section 2.

1.3 Research questions

This thesis answers questions 1-7 in the list below as a step towards the larger goal of automation in product design and manufacturing of ETO products see *I. grand scheme* in Figure 1. Previous work was done by an earlier Masters student working on the Knowledge-building project [2]. He successfully achieved a system that automatically generated robot code based on sketches made manually in NX Siemens. Remaining gaps on the path towards full automation are highlighted in *III. Thesis contribution* in Figure 1: they include the automatic placement, retrieval and exportation of weld lines from a STEP-file of an assembly where welds are not indicated beforehand – the problem description of this thesis. The focus for this work can be summarized by the following seven research questions (R.Q.s):

- R.Q. 1. How does automatic and manual placement of welds compare?
- R.Q. 2. Is design of welds repetitive enough to be automated?
- R.Q. 3. Can engineering knowledge of weld locations be expressed in logic terms in code?
- R.Q. 4. What is the best way to transfer knowledge between platforms in development of a product that is to be welded by a robot?
- R.Q. 5. What is the best way to mark a weld path in CAD, with respect to transfer to software for robot code generation?
- R.Q. 6. Which software tools are available for solving the present research problem, and what are their capabilities?
- R.Q. 7. How can programmatic CAD be made easier?

1.4 Structure of thesis

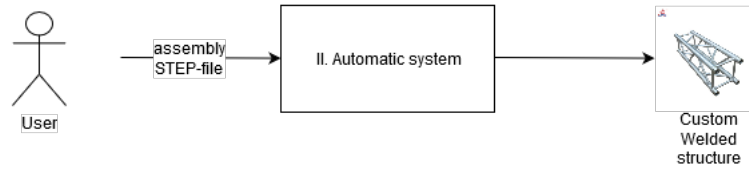
This thesis is structured in the following chapters:

- State of the art 2 chapter, on programmatic CAD and ETO, including a discussion of human and robotic welding. This chapter places this work in the appropriate research fields and opens with a figure describing the structure of the chapter. It concludes with a summary which lists gaps in research.
- Methodology 3 chapter includes an abstract approach to solving the research problem laid

out in the preceding chapter.

- Implementation 4 chapter documents how the problem was approached and which choices were made along the way. It opens with a figure describing the structure of the chapter.
- Results 5 chapter presents the subtasks that were completed towards developing a complete system. It also includes a section describing suggested testing for a completed system.
- Discussion 6 addresses the R.Q.s from the introduction and reviews the present work.
- The Conclusion 7 chapter summarizes the work, states its contribution, and lists opportunities for future work.
- Appendix A includes listings referenced in the Results chapter, and Appendix B presents progress towards publishing findings presented here.

I. Grand-scheme



II. Automatic system

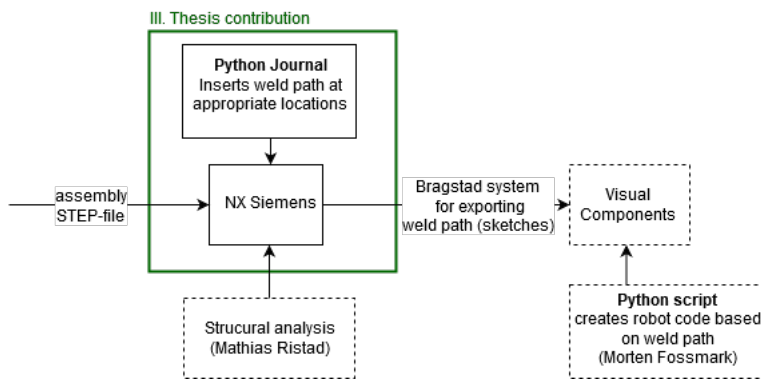
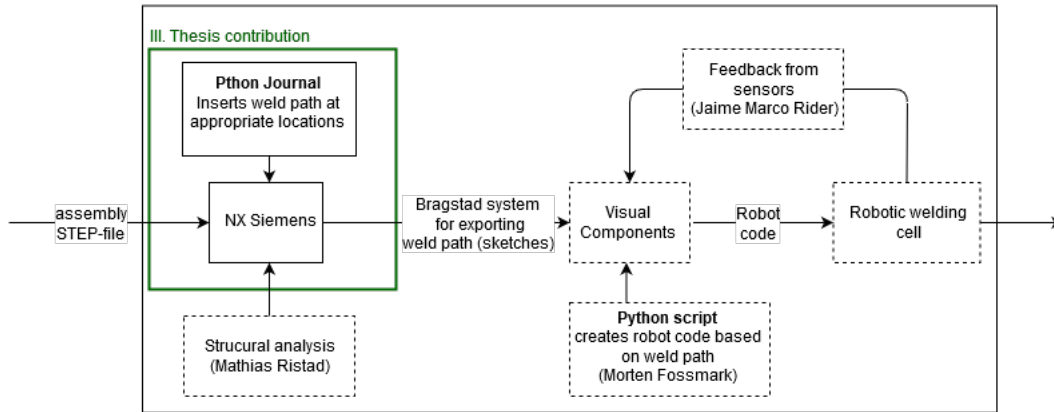


Figure 1: Thesis scope

2 State of the art

The structure of this chapter is outlined in Figure 2, which shows how the theory presented is interconnected.

The chapter is divided into three main sections: Programmatic CAD and CAD APIs 2.1, ETO products and companies 2.2 and finally a Summary section 2.3. Taken together these three sections place this thesis in the broad field of automation of design and production.

The first section describes current use of code to manipulate a CAD model and how this thesis extends the state of the art. The section opens with an explanation of how CAD is used programmatically through an API. Subsections on three different topics illustrate initial motivation for using CAD programmatically: Knowledge-based Engineering 2.1.1, Platform Independence 2.1.2 and Industrie 4.0 2.1.3.

The second section describes the domain in which the solution offered by this thesis is applicable. It opens with an introduction to the concept of ETO companies and products. Two subsections present current welding practices at these types of companies, the state of the art of robotic welding, and how the former may adapted to new technology. Marine Aluminium and Leirvik both intend to introduce robotic welding within the course of the four-year Knowledge-building project.

The final section summarizes the chapter, including defining the role of this thesis in filling a gap in existing theory.

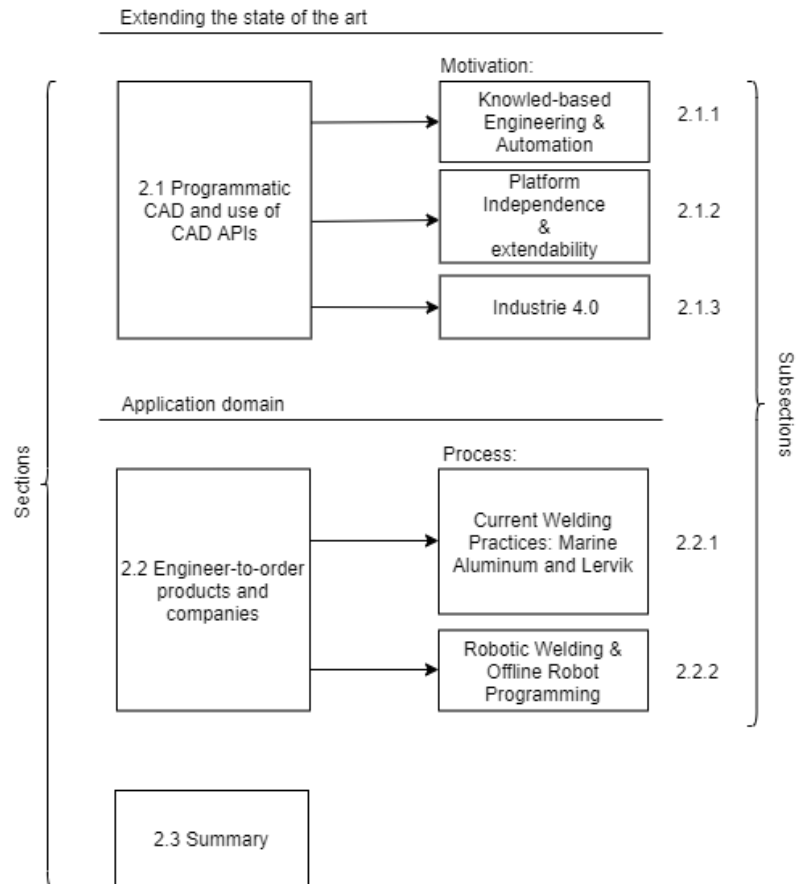


Figure 2: Structure of state of the art chapter

2.1 Programmatic CAD & CAD-APIs

This section introduces the concept of using CAD programmatically through an API. Three subsections on topics that serve as motivation for using CAD this way are presented.

CAD and code – programmatic modeling and rules. CAD models are typically constructed by sketching and generating features in a graphic environment – an *interactive* use of CAD. This is an intuitive way of creating a model and bringing an idea for a design to life. A second and less intuitive approach to using CAD is based on code – *programmatic* use of CAD. A CAD model can be constructed and edited by typing code and later visualized by running that code, as opposed to interacting directly with the graphic display. Programmatic use of CAD may seem like a detour, but can offer major benefits. One advantage is the potential for implementing rules. Rules are knowledge and requirements concerning a model, which are written and stored digitally, typically as code. *"Basically, the rule concept is grounded upon the IF-THEN-ELSE-notation known from software development. Rules are fired procedurally and can be used to execute subordinate rules or delete them temporarily from the working memory"*[4]. Rules are defined relationships between parameters and values, which can be fixed or parametric. Obvious applications involve geometric and dimensional relationships, but rules can be applied to other types of parameters as well. Vajna distinguishes between four categories of parameters:

- geometrical parameters, e.g. dimensions and position constraints
- topological parameters, e.g. suppression state of a feature
- physical parameters, values for physical properties e.g. weight
- process or technological parameters, restrictions dictated by production method, e.g. minimum radius dictated by welding tool

[5]

Parametric design is one method by which a model can be generated through conditional statements, rather than set values. A driving parameter is used to dictate the size of driven parameters. In this way, a model can be updated by a designer when subject to change in one or more driving parameters; and there is no need to design from scratch with every little change.

CAD-API definition and examples of usage. There is a host of CAD software available. Popular brands like SolidWorks, Fusion360 and NX Siemens have similar user interfaces and capabilities well suited to building models in a graphic environment. One factors that can distinguish CAD programs and which is relevant to this thesis is the software's APIs. These are key in the automation system developed in this thesis. Section 2.2 details how our industry partners define welds in their CAD models in their respective CAD software. This section defines CAD-APIs and explains current usage. NX Siemens was the CAD software used in this work and is therefore discussed first and more thoroughly than other software. Other software is mentioned in order to provide a better overview of what is available and what has been done by way of automation and knowledge transfer with respect to CAD.

An application programming interface (API) enables software interaction through a program-

ming language. The concept can be deduced from the name – an interface to an application which uses programming. Solidworks has an API that allows you to use Visual Basic for Application (VBA), VB.NET, Visual C#, Visual C++ 6.0 and Visual C++/CLI to write programs and customize your SolidWorks experience [6]. Autodesk Inventor gives users access to Inventor's functionality through Visual C++, C#, Delphi, Python and Java [7]. NX Siemens has a collection of APIs under the name NX Open. This work uses NX Open with Python.

NX Siemens APIs: Knowledge Fusion and NX Open. NX Siemens comes with the modelling language Knowledge Fusion (KF) for adding rules. It is also possible to use KF to interface between CAD and other software. KF can be used through the KF development environment known as Interactive Class Editor (ICE) or by writing a KF program in any editor and saving it with a .dfa extension. An order may be placed for custom configuration of a product and the rules in place will partially or fully determine the outcome of the order. *"The overall goal is to transform a design problem into a configuration problem using, e.g., the link to dimensioning or calculation formulae, design rules or manufacturing restrictions."*[4]. KF is a modeling language developed specifically for employing rules. As described in the Siemens documentation pages: *"Knowledge Fusion is an interpreted, object-oriented, language that allows you to add engineering knowledge to a task by creating rules which are the basic building blocks of the language."*[8]. It was initially intended to use KF for this work, but the collection of APIs known as NX Open proved to be a better option because of more extensive documentation and greater availability of learning resources. As described in NX Siemens' documentation pages: *"the NX Open suite of products provides an extensive and flexible environment for automating and customizing NX."*[9]. NX PLM Software is working to provide *"a uniform object model across all languages, included Knowledge Fusion"* [9].

NX Open is divided into Common APIs and Classic APIs. The common API is a set of languages that *"share a common object model"* and thus have the same capabilities. One can choose a programming language within the Common API based on personal preference without concern for functionality. Common API includes NX Open for: .NET, Java, C++ and Python. There are three more APIs that are part of NX Open which are not Common API. These are the Classic APIs, developed prior to the Common APIs, maintained, but not developed further by Siemens. The Siemens documentation pages have reference guides for all languages offered through NX Open.

Graphic/interactive environment and journaling. A major advantage conferred by NX Open, especially for those who lack coding experience, is the ability to record a *journal*. NX Siemens' graphic interface allows a user to automatically record a journal, a script of his or her activity, while using CAD interactively. In other words, one can use CAD, the way it is typically used, while automatically generating a script that, if played back unedited, will repeat the recorded actions. Journals are used to automate tasks in CAD and can be run as-is, or edited relative to a recording. It is possible to add custom buttons to a ribbon in the interactive environment that automatically runs a journal file or a script. Another reason for creating a journal is to work backwards from the graphic interactive environment and uncover the code generated "behind the scenes". This allows a user familiar with interactive CAD to see which programming classes and functions correspond to familiar functionality from the interactive environment. NX allows a user to record a journal in

C#, C++, Java, Python and Visual Basic. Siemens PLM Software is working to make journaling available for all languages.

This thesis extends the state of the art of programmatic CAD by designing a program for use in robotic welding. The program is written in Python and uses existing classes and functions to automatically define a welding trajectory in a NX Siemens assembly model. The following two subsections discuss Knowledge-based engineering, platform independence and Industrie 4.0, respectively. All three topics provide motivation for extending the state of the art of programmatic CAD.

2.1.1 Knowledge-based engineering and automation

The continuum of automation. Depending on the product, automation may be utilized in various phases of product development and to a variable extent within each phase. At the extreme end of the scale, a customer on placing an order would automatically generate a model with accompanying documentation, check engineering standards and produce machining code which would be sent to the workshop and result in a finished product. Alternatively, automation may be realized moderately, applied to selected parts of the product development cycle – for example limited to the production stage or parts of production. In the 2008 paper, *Levels of Automation in Manufacturing*, Frohm et al report the results of a literature study which investigated how different levels of automation are defined and concluded that automation exists on a continuum: "*Based on our review, it can be noted that automation is not all or nothing but should rather be seen as a continuum of automation levels, from the lowest level of fully manual performance (based on the capabilities of the human) to the highest level of full automation (without any human involvement).*" [10]. What is meant by automation in a given scenario will vary, this is important to keep in mind when discussing or developing automated systems. Frohm et al further distinguish between *physical/mechanical* and *cognitive/information-related* levels of automation: "*It thus seems to be appropriate to separate the level of automation into a) physical tasks, such as manufacturing technologies, and b) control tasks, such as supervision and problem solving*" [10]. The latter, cognitive/information-related level of automation, is closely related to Knowledge-based engineering which is discussed later in this section.

Parametric design and configuration. Parametric design is a way of constructing a model that updates the whole model accordingly after the change of one or more dimensions. This is achieved by defining parameters in relation to each other as opposed to assigning fixed values. Parametric design is convenient for products that are offered in different versions or in a design process where some parameters and relationships are decided but the product as a whole is still under development. Similarly, a company may offer variation in terms of configuration, e.g. a phone with the option of different amounts of storage or a car with or without a sunroof. When a product is offered in different versions, this has consequences not only for design, but across the product development cycle. When a model is altered, automatic update of the analyses of the new model and production details are desirable as well. This thesis focuses on the latter. The efforts by another student from this Knowledge-building project focuses on the former [11]. His project work uses NX Siemens, Excel and Python to automatically update strength analysis for varying sizes of a large

aluminum node. Central to the discussion is the reduction of lead time by automatic transfer of knowledge between variations or configurations of a product.

Automation and customization. Product customization adds complexity to automation. If a product changes physically, the automation of production must adapt. Frohm et al argue that it is not always beneficial to automate excessively. They maintain that "*both advanced technical systems and skilled human workers are necessary to achieve flexible and efficient manufacturing.*" [10] and that an increase in automation might degrade operator performance "[...] *degraded operator performance may be caused by e.g. lack of knowledge, gradual loss of special working skills, degradation of situation awareness.*" [10]. However, automation of products that come in variations is a popular pursuit. Companies like Engineering Intent, a provider of automation tools and services, are working specifically towards automation of complex and configurable products [12].

Engineering Intent's website has a page dedicated to defining *engineering automation tools* without referring to specific software. Some definitions from this page are included here to establish a link between automation and Knowledge-based engineering which is discussed shortly. A description of engineering automation tools offered by the website is as follows: "*These tools are unique because they incorporate configuration capabilities together with an ability to automate engineering calculations as well as geometric and spatial relationships, all in a single, integrated representation.*"[12]. In other words, engineering automation tools enable automatic updates of more than just a CAD model. Calculations, documentation and production information can also be made to automatically adapt to changes in an order. Engineering Intent further describes engineering automation tools to be "*Like a spreadsheet on steroids*"[12] in the sense that a spreadsheet has the automation advantages of programming but does not require that the user has programming skills. The list of selected bullet points below describes what an engineering automation tool is and is in keeping with the aims of this thesis.

Engineering automation tool systems..:

- *Object-oriented development environments that are specifically armed to handle the complex interrelationships and dependencies of engineering design*
- *Capable of developing proposals, quotations, spreadsheets and the like*
- *Spatial and geometric generators that can translate into fully automated CAD output, CAM data, and CAE manipulation*
- *Capture engineering rules*
- *Automatically apply rules when required; Create data for manufacturing, drawings for proposals, documentation, and manufacturing, and much more.*
- *Engineering automation uses specialized software tools to capture and then automate engineering rules and procedures.*

[12]

Accordingly, the tool developed in this thesis can be considered an engineering automation tool. The final point on the list above reads like a definition of the next topic – Knowledge-based

engineering.

Variations in defining Knowledge-based engineering. Knowledge-based engineering (KBE) has been defined in many ways, often in vague terms [13] [14]. Key to most definitions is the capture and reuse of engineering knowledge. The knowledge being captured and the means by which it is captured will depend on the specific definition and/or specific case of KBE. Wang et al list three evolutionary phases of CAD technology, where the fourth and current phase is KBE [15]. The following quote specifies the type of knowledge that is *not* stored in the predecessors to KBE: "*The domain design principles, successful design cases, expert knowledge, and experience cannot be put into the final product models.*" [15]. In other words, CAD without KBE does not include the capture of these types of knowledge. According to the authors, this leads to duplicate work and may cause mistakes. The types of knowledge listed in the quote are examples of what *can* be captured with KBE. By capturing knowledge, for example in code, certain tasks may be automated, reducing the need for re-work.

Definitions and descriptions of KBE. Similarly to Wang et al who describe KBE as a phase in CAD technology, Chapman et al refer to KBE as an evolutionary step in Computer-aided Engineering (CAE) [15] [16]. Chapman et al further describe KBE as a system that combines CAD with object-oriented programming (OOP) and artificial intelligence (AI) to aid in design automation of custom products. The ultimate goal of KBE according to Chapman et al is to "*capture the best design practices and engineering expertise into a corporate knowledge base.*" [16]. In their 2011 paper *A critical review of Knowledge-Based Engineering: An identification of research challenges* Verhagen et al review 50 research articles on KBE in an attempt to identify theoretical foundations and research issues, and to discover why KBE is not applied more frequently. According to Verhagen et al, one of the main objectives of KBE is to reduce time and cost by automating repetitive design tasks: "*KBE is applied to automate repetitive design tasks and achieve significant design time savings, enabling designers to explore a larger part of the design space during the various design phases.*" [13]. They, too, trace its origins to a merging of AI and CAD introduced in the 1980s. They reference two examples where OOP is included in defining KBE, one of which is Chapman et al. Verhagen et al imply that KBE is to applied in the design phase, while Siemens stresses applicability by all disciplines throughout the engineering product life cycle including, but not limited, to the design phase [17]. According to Dinh, although applying KBE can be time consuming, the time saved will ultimately be higher for a product which will have several configurations: "*Through integrating KBE technology with CAD software, design engineers create virtual product configurations by applying a scripting language to the CAD model. It requires time and effort invested in a different way than traditional design method, which may cost more to develop. However it is more efficient and accurate when producing multiple configurations.*" [18]. Notice that Dinh describes KBE as an entity separate from CAD, in contrast to some earlier work where KBE is partially defined by CAD – yet a discrepancy in definitions. In practice, there is often a relationship between KBE and CAD. Several publications on KBE present cases of developing a product with specific CAD software. Different CAD software vendors offer their descriptions of KBE as well.

KBE and NX Siemens. On NX Siemens' designated *Introduction to KBE* page in their documen-

tation pages, KBE is described in this way: *"Knowledge Based Engineering (KBE) is the process of capturing and structuring reusable knowledge bases to create and enhance solutions for a product during its entire life cycle. These knowledge bases can exist in many forms such as spreadsheets, handbooks, engineering formulas, drawings and documents, or in rules of thumb that are based on human judgement. KBE is able to create and reference such knowledge bases and make them readily available as an aid to the engineering process through the use of computer assistance and software tools."*[17]. In short, the focus in this definition is the interaction between software tools and knowledge bases in various forms. On the same page, they point out what KBE adds to CAD technology:

KBE is the key to being able to answer questions of some significance that traditional CAD systems to date have not been capable of addressing such as:

- *"What was the rationale behind this design?"*
- *"Have any design constraints been violated?"*
- *"How much will this product cost?"*
- *"Can this part be manufactured?"*
- *"Will this part meet its performance goals?"*
- *"Is this design optimum or are there better alternatives?"*

[17]

KBE modeling languages. There are different means of applying rules to a CAD model. KBE systems, dedicated software tools for KBE, are now built into CAD software [18]. These systems allow for the digital capture and re-use of knowledge, for example as rules. Operations triggered by rules represent tasks where human involvement is removed, effectively automating those tasks. There are modeling languages created specifically for the use of rule-based design. NX Siemens has KF, which in Siemens' documentation pages is referred to as an API, *"an automation tool"* and a *"KBE technology"* [8]. Siemens states that rules are fundamental to KBE. They describe rules as defined input-output relationships, which are able to access knowledge bases [17]. According to Gembarski et al *"Implementing design rules in any given CAD software typically requires an extension, such as Knowledge Fusion in NX Siemens or iLogic in Autodesk Inventor, whose purpose is implementing design-rules."*[4]. However, any available API is a potential medium for adding rules, as will be illustrated in the next paragraph. Inventor's iLogic is another example of a KBE-centered modeling language: *"The iLogic programming language is similar to script languages. Common constructs like if-then-else or select-case decision trees, while loops, the use of sub procedures and a class concept are usable. As command library the snippets include code templates for almost every modeling context within Inventor."* [5]. These languages are intended for use by engineers and don't require programming expertise [18]. Cases of KBE modeling languages being applied are listed at the end of this subsection.

KF vs NX Open. Both KF and NX Open are referred to as *toolkits* in the NX documentation pages [9]. Both options were considered for use in this work before landing on NXOpen. KF has far fewer learning resources and less documentation available than major programming languages such as those offered by NX Open. A language does not have to be created for the purpose of rule-

based design in order to be used to create rules. KF was created for rule-based design, but it is possible to enforce rules by means of more general purpose languages. In discussing the differences between KF and NX Open on an engineering forum, system developer Graham Inchley wrote in 2014: "*NXOpen is very powerful, but only executes outside the NX update mechanism. So as the code creates or changes something, it then has to wait for NX to update the part. KF on the other hand is a 'rule based' language that interacts directly with NX's update mechanism. This means it can do things that NXOpen cannot.*" [19]. Conversely, in a 2018 post to a KF discussion on Siemens' own discussion board, Taylor Anderson wrote "*For the very best custom feature behavior, the NX Open API is now the very best method.*" [20]. The use of one does not exclude the use of the other as pointed out in the NXOpen programmers guide: "*NX provides a comprehensive set of automation tools including Knowledge Fusion and UI Styler. [...] This guide only provides overviews of how NX Open works with these other tools.*" [9].

Below are some of the reasons for using NX Open as formulated by Siemens:

- *Ability to customize NX to meet your specific industry and process needs*
- *Decreased time to market by automating complex and repetitive tasks*
- *Reduce rework by capturing and reusing company and industry best practices*

[9]

Although these three points describe NX Open the last two describe KBE equally well. The quoted list underscores that using NX Open for KBE is not a radical idea.

MOKA framework for applying KBE. KBE aims to reduce lead time. However, creating a KBE system demands a certain "activation energy". Naturally, the time saved from using KBE should ultimately outweigh the cost in time of employing KBE. Part of the MOKA method for creating a KBE system is to assess whether KBE is advantageous in a given scenario. MOKA stands for *methodology and software tools oriented to knowledge-based engineering applications*. MOKA is meant to aid in planning and organizing KBE applications [21]. It is comprised of six steps: identify, justify, capture, formalize, package and activate. The capture and formalize steps are the most elaborate [22]. As described by Golkar, the formalize step is where knowledge takes a standard defined form: "*under formalize the formal model is created from the informal model. The language that is used to model is called MML (MOKA modelling language) and it's based on UML (Universal modeling language).*" [23]. MOKA was developed in an effort to achieve a common methodology for performing each of the six steps and applying KBE [21]. Siemens has a similar less formal description of how a KBE system can evolve:

- *Identify what knowledge is reusable.*
- *Gather knowledge data from the various sources.*
- *Format the knowledge data into rules constituting a knowledge-base.*
- *Use the Rules from this knowledge base in conjunction with computer aid (knowledge Engine) to come up with new solutions and decisions.*

[9]

Methodologies for KBE provide guidance for its application, and can be employed loosely or strictly.

KBE past research cases. Gembarski et al delineate different approaches to creating a KBE model in Autodesk Inventor [4]. They describe a skeleton approach where parameters are linked using the parameter dialog box, and compare it to a rule-based approach using Inventor's iLogic language to formulate rules. They offer three examples using various approaches: a base frame, a crank and a platform. The base frame is used at manufacturing stations and is modeled using iLogic rules. A change in the dimensions of a mounting plate for the base frame assembly will update the entire assembly and accompanying drawings. This is said to have reduced design and documentation time from hours to minutes. The crank uses a skeleton model with an embedded spreadsheet. Spreadsheet input consists of standard part sizes and geometric boundaries, and calculations are returned as a strength report. iLogic rules were also used, for updating the geometric model in this example. Their final example is a platform consisting of predefined modular building blocks for the platform frame, hand-rails and stairs (respectively). The example uses excel to configure a platform and check that it adheres to existing standards, as well a skeleton model for calculations and iLogic rules to update geometry.

2.1.2 Platform independence and extendability

In their paper on ontology, platform independence and KBE, Sanya et al define platform independence as follows: "*In software engineering, a platform-independent knowledge model is a model of a business or software system that is independent of the specific technological platform used to implement it.*"[24]. The major automotive and aerospace industries have progressed the furthest in terms of successful application of KBE [13]. Still there is progress to be made in these industries as well, for instance with the issue of platform dependence: "*[...] the design and implementation of multidisciplinary KBE systems in the aerospace sector are usually platform specific and domain dependent. Due to the nature of specificity within KBE systems, knowledge reuse becomes an issue because the 'design' and 'implementation' of KBE systems are often tied down to a specific technological platform. [...] Therefore, if there were to be a platform change, this will mean a complete re-write of the KBE software system.*" [24]. Platform independence is significant for re-use of the KBE system itself. In other words, if a KBE system is platform independent, then the system's application can potentially extend beyond the single case for which it was designed. Cho et al reiterate the issue: "*While knowledge acquisition is still a bottleneck process, the formalized engineering knowledge is still too often encapsulated in CAD models and in KBE systems developed in vendor-specific environments.*"[25]. In attempt to overcome this difficulty they developed a solution which uses *Multi-CAD* to unify information from CAD models made in different software: "*the proposed solution combines the use of a Multi-CAD API library – which allows platform-independent and automatic extraction of engineering knowledge from CAD models into an XML-based representation*"[25]. Their goal was to create a system that could extract data and design-intent from CAD, store it in a XML document and from there be able to apply it to KBE and PLM systems regardless of CAD, KBE or PLM platform. Their system is dependent on CAD vendors providing sufficient functionality with their respective APIs. Another disadvantage is the need to develop multiple interfaces [25]. Lin et al put the need for a common language in

laymen's terms: "A standardized terminology needs to be semantically consistent across organization boundaries, since the communication aspects of information require that communicating parties have the same understanding of the meaning of the exchanged information. This assumption is simple: if everyone adopts the same concepts, vocabulary, and language, any data expressed within this language will be accessible to everyone. For example, technical standards for product information and CAD/CAM documents have been realized by efforts like Product Data Management, Product Lifecycle Management and the Standard for the Exchange of Product Model Data—STEP." [26]. They use ontology to "[...] facilitate communication and information exchange in inter-enterprise, multi-disciplinary engineering design teams [...]" [26]. Ontology in aid of platform independence is discussed next.

Ontology and AML. This thesis uses the NX Open API, but there are alternative and supplementary means of creating a KBE-system that may hold more promise for integration of software and extendibility. Two of these are introduced briefly here to facilitate an understanding of the state of the art of platform independence and the significance of programmatic CAD in this regard. Note that this discussion focuses on the application and benefits of these modeling languages and does not provide details of how they work. The World Wide Web Consortium (W3C) states the following on ontologies compared to other ways of storing and sharing information: "Ontologies are critical for applications that want to search across or merge information from diverse communities. Although XML DTDs and XML Schemas are sufficient for exchanging data between parties who have agreed to definitions beforehand, their lack of semantics prevent machines from reliably performing this task given new XML vocabularies. The same term may be used with (sometimes subtle) different meaning in different contexts, and different terms may be used for items that have the same meaning." [27]. Ontologies are used in many domains, including healthcare and web development. There are several research cases using ontology for product development [28]. In the 2007 paper, *A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration*, Lin et al wrote the following in support of ontology: "There are many potential application areas for this approach since companies enter into temporary inter-enterprise collaborations for many types of business ventures and consequently many different types of information may need to be exchanged or shared. For example, details of products or components at different stages of design or manufacture, or details of available manufacturing facilities or resources, etc." [26].

Another language aimed towards platform independence is AML, described thus by its developers at Technosoft Inc.: "Adaptive Modeling Language(AML) is an object-oriented, knowledge-based engineering modeling framework. AML enables multidisciplinary modeling and integration of the entire product and process development cycle." [29]. An example of AML is found in the aerospace industry where the Wright Patterson Airforce Base Research Laboratory discusses: "[...] the application of the Adaptive Modeling Language (AML), developed by Technosoft Inc., to the Trans-Atmospheric Vehicle design process. Integration of several design tools into a single unified environment using AML is currently under development at Wright Patterson Air Force Base. The resulting design process promises to significantly reduce design time. Additionally the environment will allow real-time collaborative design over the Internet." [30].

Programmatic CAD and platform independence. Creating a system that can transcend specific

software and even domains would be a great feat in terms of both human and digital communication. Attempts have been made at using CAD APIs to extract, export, and translate model information to a standard form [14]. Reijnders echoes the need for interpreting knowledge across platforms in regard to KBE: "*KBE applications are dependent on knowledge, but they can only work with knowledge that is structured in a way that these applications can understand. Since not all knowledge is structured in the same way and/or accessible [...], we are again back at the knowledge structuring problem [...].*"[14]. Using CAD programmatically can be a step in achieving platform independence, but does not solve the challenge altogether. Just as a KBE solution may be specific to certain software, it may also be specific to certain programming languages. This is discussed in the discussion chapter 6, with respect to the results in this thesis.

2.1.3 Industrie 4.0

Industrie 4.0 – the fourth industrial revolution. Industrie 4.0 (I4.0) refers to the fourth and ongoing industrial revolution. In brief, the prior three revolutions were marked by: first, the introduction of mechanical systems and subsequent centralizing of production (the steam engine driven mechanical loom); second, the division of labor and assembly lines (at a slaughter house in Cincinnati, followed by T-ford automobile production); and third, automation through programmatic logic control (PLS) [31]. I4.0 involves using technology that is already available in new ways. Drath et al write: "*Industrie 4.0 is the triad of physical objects, their virtual representation and services, and applications on top of those.*"[31]. I4.0 requires transfer of information between entities, such as those listed in the previous quote, across physical and digital borders. The result is an industry that uses the internet and large knowledge bases to enhance performance. I4.0 integrates information by means of the Internet of things (IoT), internet of services (IoS) and internet of people (IoP) [32]. An example is found in the shipping industry, where sensors have been added to the motor of a harbor crane to sense the real-time actual weight of shipping containers. Actual weight deviates from documented weight. The weight data from the crane-motor are sent to a digital twin of the cargo ship. Here the optimal placement of the containers to achieve weight balance is calculated. This information is sent back to the motor which then expedites ideal placement. Optimal placement of shipping containers on cargo ships reduced fuel consumption significantly, saving up to 1000 US\$ per day per ship on fuel expenses [33].

Industrie 4.0 was introduced by Germans at the 2011 Hanover Fair, hence the German spelling of "industry". The phrase has caught on globally and is frequently used in advertisements. The fourth industrial revolution is in progress, but not yet in full swing [31]. The concept continues to be refined as standards are developed for classifying and navigating I4.0. One of these is the Reference Architecture Model Industry 4.0 model (RAMI 4.0) [31]. RAMI 4.0 is a 3D model, incorporating the three dimensions that make up I4.0, see Figure 3. The horizontal axes are derived from existing standards, specifically the IEC 62890 standard for life-cycle management for systems and products used in industrial-process measurement, control and automation (left horizontal axis) and IEC 62264 & IEC 61512 standards for enterprise-control system integration & batch control (right horizontal axis) [34]. The vertical layers from bottom-up represent: asset, integration, communication,

information, functional and business, see Figure 4. The model allows for mapping of a company or product in terms of I4.0. The model also serves as a communication aid, promoting a focused conversation on the multidimensional topic of I4.0. It is presented here to emphasize the convenience of having information available in code to enable transfer between layers.

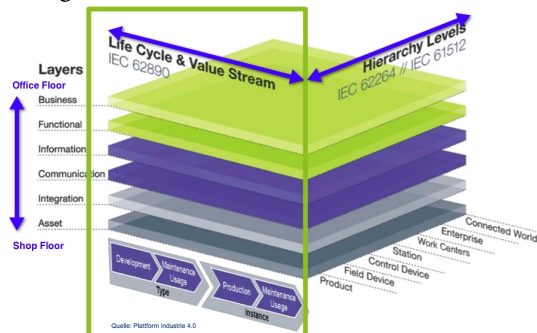


Figure 3: RAMI [1]



Figure 4: RAMI vertical layers [1]

Industrie 4.0 and programmatic CAD. The following quotes refer to KF, but are included to emphasize the link between programmatic CAD in general and I4.0. Siemens' documentation pages state that: *"Knowledge Fusion (KF) - This API is an interpreted, object-oriented, language that is embedded in NX. [...] the language has the capability to access external knowledge bases such as databases or spreadsheets and to interface to other applications such as analysis and optimization packages."*[8]. The programmatic format enabled by an API (KF or other) lets a user control a model with the flexibility and efficiency that comes with programming, using if/else-statements, loops, etc. Furthermore, information stored in code can be operated on and processed by all the functions and classes available in the language used, including tools that may not be internally available in the CAD software. For example, a rule can be made in CAD using advanced programming functions for math or statistics that are not available internally.

The core of this thesis is how data, information and knowledge are stored, transferred, analyzed and processed. This relates back to the comparison of programmatic and interactive ways of using CAD. Not only does the programmatic approach cater to KBE by means of capture and re-use of knowledge in code, it also expands the field from which information is gathered. Thus the programmatic approach can accommodate KBE as well as I4.0.

2.2 Engineer-to-order and small-medium enterprises

This section opens with a general introduction of the engineer-to-order (ETO) concept. The two subsections discuss welding in the context of ETO companies. Subsection 2.2.1 presents current practices, challenges and future plans in terms of welding at Marine Aluminium and Leirvik Aluminium, partners of the Knowledge-building project. Neither company uses robots for welding currently. The second and final subsection 2.2.2 presents the state of the art of robotic welding and adoption of this technology by ETO companies.

Characteristic of an ETO product is some variation that is known only once an order is placed and a configuration is selected. The uniqueness of a product adds to the challenge of automation: "To date, ETO companies struggle with defining the most appropriate degree of standardization and automation." [35]. Willner et al argue that ETO-products can be divided into four archetypes and that by classifying a given ETO product in one of these four, it is easier to estimate the obtainable amount of standardization and automation. The degree of engineering complexity combined with the number of units sold annually, determines the archetype of an ETO-product. This thesis is built around large welded aluminum structures, e.g. bridges, helipads, living quarters or ship hulls. These structures would be classified as Quadrant I type ETO-products in the Willner et al system:

"Quadrant I: Complex ETO. Traditional "one-of-a-kind" products that are ordered in low volumes (750 units per year) and contain a high-engineering complexity (more than or equal to 2,000 hours per unit) are located in this quadrant. This quadrant corresponds with the definition of ETO most commonly used in the literature (Hickset al., 2001; Olhager,2003; Gosling and Naim, 2009). Examples of products ideally positioned in this quadrant are ships, oil platforms, or nuclear plants. Products located in this quadrant require a high-engineering effort since they are engineered according to precise customer specifications. Initially, only rough product concepts exist, and these are finalized according to the specifications of precise orders (Maffinet al., 1995). Lead times for this type of products are long with order-specific engineering activities contributing to a substantial amount to the total lead time (Littleet al., 2000; Pandit and Zhu, 2007" [35].

The challenge inherent in developing an ETO product is responding to a custom order quickly and at low cost [36]. If applied successfully, KBE could help achieve just that as its aim is to: "Reduce lead-time (by capturing and re-using product and process knowledge)" [22]. As mentioned in the previous section, under Platform Independence and Extendibility 2.1.2 ontologies have been used in product development. One example is by Yujun et al who created a KBE system for custom development of a configurable rotor: "The rotor customized according to customer needs is the key component of industrial steam turbine and belongs to typical ETO product." [28]. The idea of reducing lead-time in ETO companies by means of KBE is not new, but its development outside of the automotive and aerospace sectors is in an early phase.

Automation, KBE and ETO. By capturing knowledge, automation is enabled, which in turn can reduce lead time, a major KBE objective. Even in production of non-identical products, some applied knowledge may be eligible for automation. KBE is especially interesting in the context of ETO companies given the inevitable variation in their products. If engineering knowledge can be

captured in these companies, it may save an engineer from re-doing design, analyses or even production work from one product iteration to the next. Definitions vary, but most commonly a product is classified as ETO if its production is triggered by an order, i.e. the product is not produced unless ordered [35]. This is because some degree of customization is communicated in the ordering of an ETO product. The product design is based on the configuration requested at the time of the order [28]. Consequently, design is a main contributor to long lead times in development of an ETO product [37]. KBE can remedy this. Willner et al describe KBE as synonymous with design automation [37]. Because ETO products are subject to variation, implementing automation is more challenging than with a highly standardized mass production type of product. In dealing with ETO, every stage of product development must adapt to whatever configuration is defined by an order. Despite variations between potential outcomes of an ETO product, some design elements and engineering tasks often remain constant. If these tasks can be captured as rules in code, re-design for each configuration can to some degree happen automatically.

2.2.1 Current welding practices and challenges

In a 2019 conference presentation Egeland expressed Norwegian industry needs: "*Norwegian industry requires rapid changeover between different product variants*"[38] and "*Advanced robotic systems that are profitable for small series.*"[38]. ETO companies making "*Advanced products of high added value*" [38] are typical of Norwegian production industry. Norwegian companies that fit this description are Leirvik and Marine Aluminium.

Leirvik was established in 1946. The company is located off the western coast of Norway and employed 300 people as of 2019. They engineer and produce on- and offshore products, including but not limited to, walkway bridges, suspension bridges, helipads and living quarters for various applications e.g oil platforms. They offer both standard and made-to-order products. Leirvik intends to acquire a Fanuc welding robot within the period of this Knowledge-building project. They have provided CAD models of a bridge element, a living quarters corner, a main column and a truss frame construction, all of which are eligible products for venturing into robotic welding. They are considering using the main column for a robotic welding pilot-project. [39]

Marine Aluminium (M.A.) was established in 1953. It is a global company with offices in Korea, Brazil and beyond. They engineer and produce helipads, stairways, gangways and other large aluminum products. The company is located about four kilometers in bird fly distance from Leirvik, where offices and production facilities are co-located. Elements of their production capabilities include a friction stir welder, a large water jet cutter and highly skilled human aluminum welders. They have provided CAD models of large nodes for joining beams and bars in helipads. The node stretches roughly 1m x 1m and is eligible for robotic welding. They too, intend to acquire a welding robot within the duration of the Knowledge-building project. [40]

Defining welds in CAD. In response to defining welds in their CAD models, Geir Mosaker, lead structural engineer at Leirvik writes by email: "*In the models I've sent you, the welds are only indicated with leaders on the drawings, no modelling of welds have been done in any projects yet.*". They use Autodesk Inventor for their projects. M.A. also uses Autodesk Inventor. To define welds

M.A. uses a custom application that places a hidden feature on a line to indicate where a weld should go linearly end-to-end. The application will be available to all in a new release of Inventor. To place a weld-feature like this in a CAD assembly, each line needs to be selected individually and manually. The engineers confirmed the usefulness of potentially speeding up this process.

There are different ways of indicating weld locations in a CAD model. In the past these weld indicators have not been made with considerations for robotic welding as the job has been done by human welders. The intention has not been to define robot trajectory, but rather to communicate with the human welder. This calls for a reconsideration of weld definition in CAD in preparation for robotic path generation. Even without considering robotic welding, being able to accurately identify and mark welds in a large CAD model with some degree of automation would satisfy an industry need. Marine Aluminium spends a significant amount of time on placing each weld in a CAD model when time is of the essence.

2.2.2 Robotic welding

Welding aluminum comes with added challenges compared to welding other metals such as steel [41]. This is due to aluminum's physical properties such as high thermal conductivity, oxide formation at surface and solubility of hydrogen in molten aluminum. These factors can contribute to porous or cracked welds. When replacing a human welder with a robotic welder, the same challenges inherent in working with aluminum must be met. In addition, deflection of the robot must be accounted for. An artisan welder knows how to compensate his or her motions to accommodate the idiosyncratic behavior of aluminum. This skill needs to somehow be adapted by welding robots. One solution has been to use sensors for measuring offset and correcting for displacement in real-time, summarized by Shen et al: *"If, as is often the case, wide fixturing tolerances, part-to-part dimensional variations, assembly inaccuracies, or in-process thermal distortions are present, some form of sensory feedback is necessary to guide the welding robot and control the welding process to produce a high-quality weld."*[42]. One contribution to advancement of robot technology has been the use of sensors as explained above. Another contribution is offline robot programming, discussed next.

Offline robot programming. With offline programming (OLP) software is used to define a robot's trajectory and generate robot code accordingly. The alternative is to use a pendant which requires interaction with the robot; however, the robot is not available to do work while being programmed. OLP can be used to generate robot code independent of the robot, i.e. while the robot is active. Tarn et al comment on pendant ("teach and playback") programming being time consuming and on deflection issues: *"At present, more and more welding robots have been applied in the automatic manufacturing process. However, most of them are the type of primary "teach and playback" robots, which must be taught in advance for plenty of time. These types of welding robots must be taught and programmed again for the different work-pieces and they cannot self-rectify a offset during the welding process."*[43]. In addition to reducing the time when robots are out of production, Wittenberg lists these benefits of OLP: *"Other benefits include improved operator safety and the quicker introduction of a new robot cell after generating a pre-installation program. Given suitable software, robot simulation, calibration and offline programming form a seamless process in*

which robot programs may be transferred to and from each stage." [44]. For these reasons, OLP is becoming increasingly popular and there are many OLP software vendors on the market. A 5-axis CNC-machine can be compared to a robot in terms of degrees of freedom: both move freely in three dimensional space within a given span. However, two major differences are programming, g-code for a CNC-mill and robot code for a robot, and stiffness. According to RoboDK CEO Albert Nubiola: "it's important to understand that a robot is not a CNC. With the right software you can make a robot behave like a 5-axis milling machine, however, robots are not as stiff or accurate as a CNC." [45]. In the same article on navigating the wide selection of OLP software, the following considerations are listed when selecting robotic simulation software:

- Will the software be compatible with the robots and tools in use?
- Is the software best suited to the process, such as arc welding or painting?
- How much robot programming or CAD experience is required to use the software?
- And how much will it cost?

[45]

They continue to present an evaluation of Siemens Process Simulate, Robotmaster, OCTOPUZ, RoboDK and Delfoi which runs on the Visual Components simulation platform. This work assumes Visual Components is to be used for robot programming. This is not critical as the majority of the work done for this thesis is on the CAD-end. When it comes to software compatibility, the beforementioned article states: "It's not uncommon for users of professional software in all industries to want to stick with one vendor in order to keep things simple and ensure compatibility. For example, if your company uses SolidWorks for CAD from Dassault Systemes, you may be inclined to choose Delmia, Dassault's robot simulation software. However, several experts we interviewed for this article did not agree with this approach." [45]. Again, choice of OLP software is not critical for this thesis, but it will be an important consideration going forward with the Knowledge-building project.

The flowchart in Figure 5 is from Hagen's thesis, showing his system for automatically generating a robot trajectory from CAD sketches. The figure has been edited: the bold arrow highlights where Hagen's solution is not automatic, and the problem definition of this thesis.

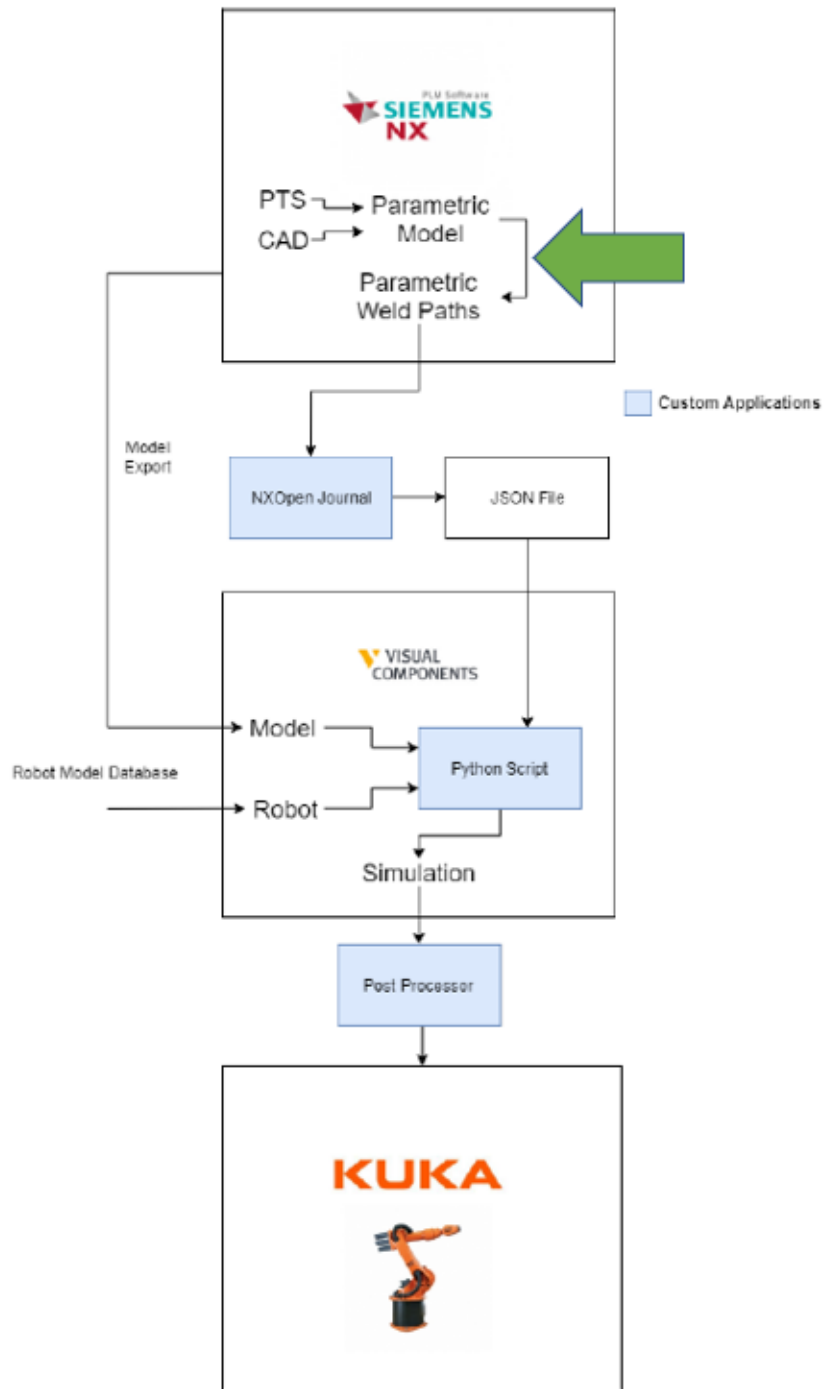


Figure 5: Hagen's solution flowchart [2]

2.3 Summary and thesis contribution to state of the art

KBE is the capture of knowledge from an engineering process, in a way which allows for re-use. This frees up time for engineers to perform creative, non-repetitive tasks. *“Knowledge-Based Engineering has held great promise since its first applications. Despite this promise and associated reported advantages to its adoption, KBE has to date not achieved a convincing breakthrough, apart from major aerospace and automotive companies.”* [13]. Many attribute the modest application of KBE to the substantial time investment involved in developing a reliable system. *“A challenge with KBE is that it takes too long to build up a design model and therefore is not suited for products that are not recurring”*[23], Verhagen et al add that: *“Notably, the KBE research field is still in development, with methodological and technological considerations constantly evolving”*[13]. Another concern is the level of programming introduced in the design and manufacturing domains, and the risk of dependence on software engineers. *“In addition, developing and maintaining these KBE tools requires software engineers. They have to work together with experts to capture the knowledge in the KBE tools, which is a slow process. Once in the KBE application, the knowledge has become opaque and inaccessible (except for the software engineers). In other words, the KBE tools are black boxes, making it impossible to manage and reuse the knowledge contained inside them.”*[14]. Although NX Siemens’ Knowledge Fusion is a modeling language intended for KBE, it has far fewer search results, articles, tutorials, etc, than well-known languages in the NXOpen collection of APIs. KBE research is often case-based evaluations of success or failure. *“Though there are KBE research papers available from various application domains, very few of these papers reflect on what is known about the KBE research field as a whole.”*[13]. Rather than develop a system based on variations of one product, this work constructs a system independent of a specific product, with intended application to a range of products – the general case of large welded aluminum structures. This will allow the KBE system *itself* to be re-used for new projects. This work designs a basic system that suggest weld locations automatically in CAD-models of large assemblies. Prior work has successfully generated robot code automatically based on sketches made manually. The manual marking of welds is a gap that must be closed to achieve full automation. The main practical contribution of this work is progress towards closing that gap.

3 Methodology

3.1 System architecture

Figure 6 below illustrates a generic simplified system for automation of robot code generation. The box labelled CAD is where this work has been focused. The APIs in both the CAD and OLP modules are utilized to generate a weld path and robot trajectory, respectively. Input to the first module is a STEP-file and user requirements, output is weld path data, which is input for the next module. Output of the OLP module is robot code. This diagram provides context for presented work. A complete system for automatic welding will include analysis and feedback in addition to what is included in Figure 6. The flowchart in the next section describes the processes that take place within the CAD and OLP modules.

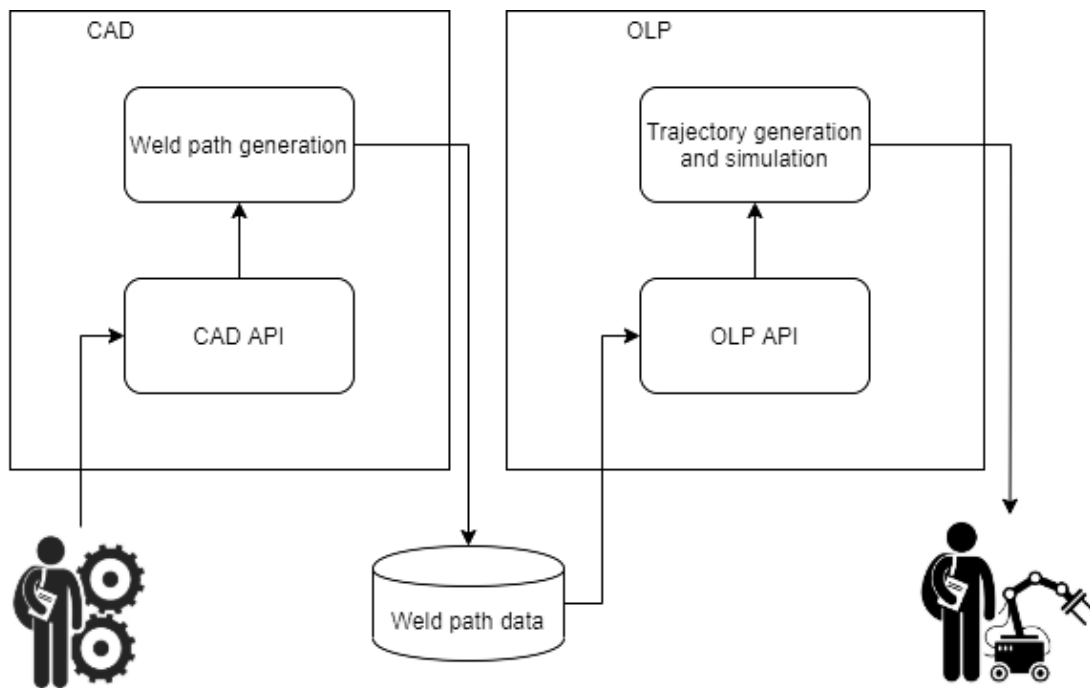


Figure 6: System architecture

3.2 Flowchart of system processes

This section is an implementation overview which includes the main modules of the system – Requirements configurator, CAD and OLP – without delving into vendor- and language-specific details. The modules and their contents are depicted in the flowchart below in Figure 7. The first module is the Requirements Configurator, this is where the engineer enters necessary model data, selects which standards are to be applied, and adds conditions specific to the case at hand. Ideally this will be done through an easy-to-use dialog box, development of which is reserved for future work 7.1. The next module, CAD, is where this thesis adds the most value. In the CAD module the system combines data from the engineer with an intersection curve builder for placing sketches which define a weld path at intersections between components, with exceptions set in the prior module. The weld path sketches are stored in a variable and exported to the required format for OLP software. The OLP module loads the weld path data in the appropriate format and generates a trajectory and corresponding robot code. For every module there are decisions to be made in regard to software, programming languages and data storage. By having the second module run on the standard model format STEP, the user is free to stick with his or her preference of CAD software. In choosing software for the CAD and OLP modules, APIs are one factor to evaluate as discussed in the state of the art chapter 2.1. The intention is for there to be a user interface that is intuitive enough for the programming language used to not matter to the user. Programming language can be selected based on the system developer’s preference and availability of the APIs of the selected software. Another decision point occurs at the intersections between the CAD and OLP modules, where weld path data is transferred. Hagen used a json file as an intermediate between CAD and OLP, as illustrated in his solution diagram 5. Alternative options include MATLAB, CSV, and XML. This work is based on making an addition to Hagen’s system as indicated by the bold arrow added to his solution diagram. Future work may involve modifying the system, for example by reconsidering software, programming languages and data storage.

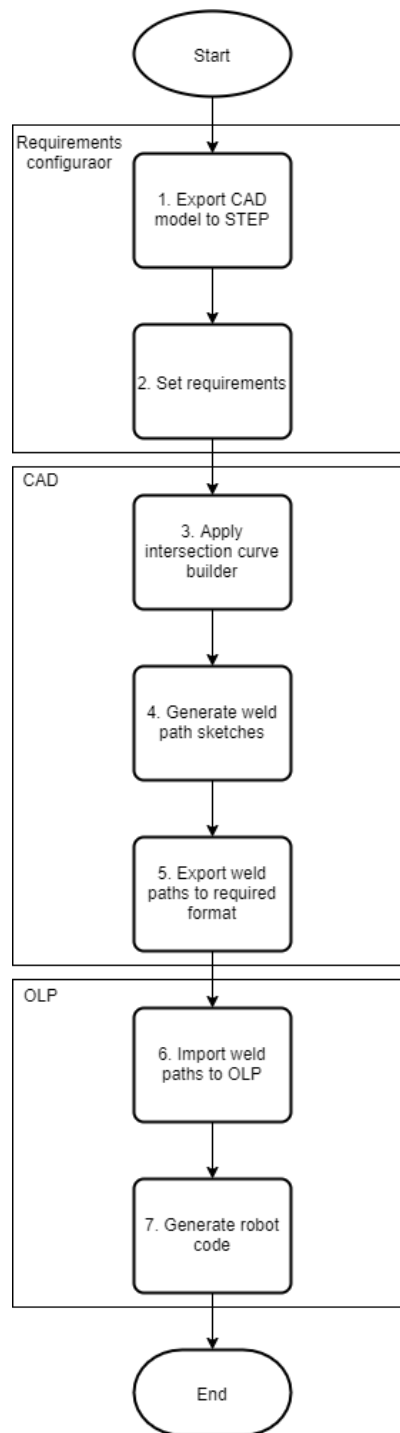


Figure 7: Flowchart showing main modules

4 Implementation

The structure of this chapter is outlined in Figure 8. The aim of this thesis is to automatically define and export a weld path from CAD. A STEP-file containing an assembly model is input into the system. The operation is divided into three main tasks: find potential weld locations, mark and store a weld path, for example with sketch lines, and export the weld path. The first section of this chapter is divided into three subsections discussing each main task in plain English. The challenges and choices associated with the respective tasks are documented, including software and language considerations.

The second section is also divided into three subsections. The first subsection uses a flowchart, with language- and software-specific details, to describe the system modules. The second subsection is a pseudocode of the system. The final subsection divides the main tasks from section 4.1 into coding subtasks. Some of these tasks have been successfully completed, some are in progress, and others remain undone. The tasks where progress is made are presented in the results chapter 5. Dividing the system into separate elements of code allows for isolated testing of each element, identification of challenges and assessment of progress towards the final goal.

This work uses NX Siemens and Visual Components for CAD and robot programming, respectively. The solution is not platform independent. The system may be adapted to accommodate other software, but cannot be directly applied. The methodology is independent of platform, but the implementation is not. This is discussed in the discussion chapter 6 and emphasized as a consideration for future work 7.1.

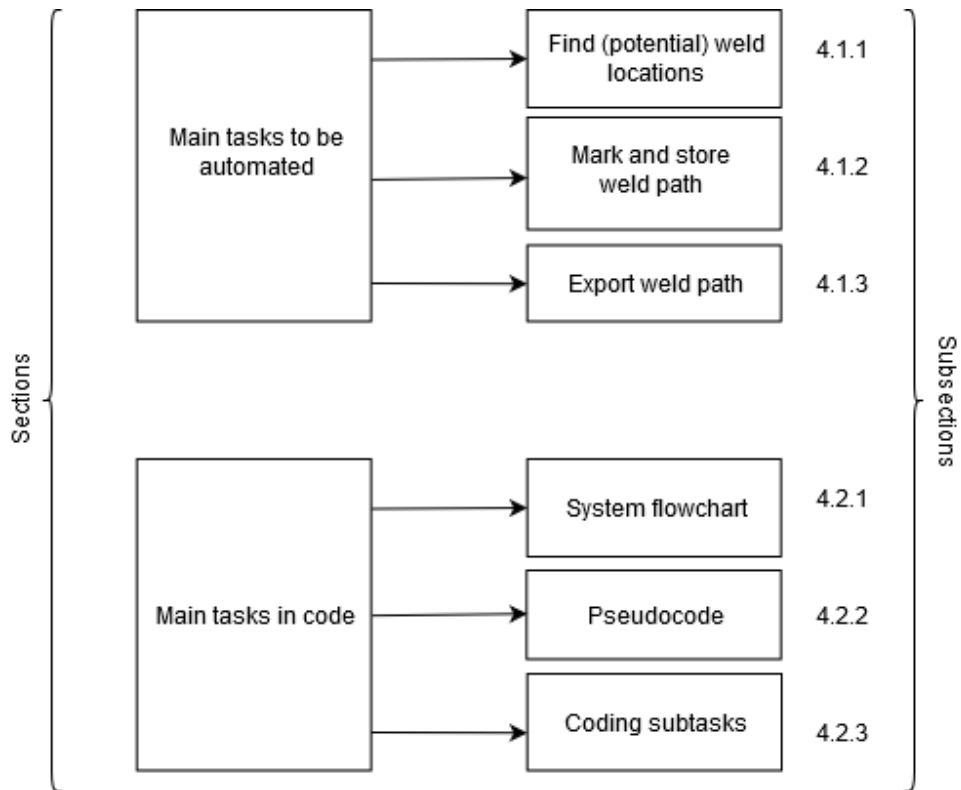


Figure 8: Structure of implementation chapter

4.1 Main tasks to be automated

4.1.1 Find potential weld locations

The system being developed here assumes that welds are located where faces from separate components are in contact. If no additional user requirements are added to the system, welds will mistakenly be located where surfaces are in contact but are not supposed to be joined. The user is to add filters to the system via rules, resulting in removal or addition of weld locations based on the engineering knowledge for a given case. An example of a rule added by the user could be to not define a weld path that is longer or shorter than a given value. Ideally, the user will have a simple interface for adding his or her conditions. Without such an interface, the user will have to write code to edit the system default of defining welds at component intersections. This work focuses on the basic system for defining a weld path where surfaces from separate components meet. The CAD models received by the project partners are in the standard format .STEP. Essentially all CAD programs can export a model to .STEP, so it is preferable that the system be able to detect welds in this type of file.

4.1.2 Mark and store weld path

The second main task is to mark the identified weld locations. There are two major aspects of this challenge:

- i. Definition of what indicates a weld line, e.g. a sketch, a set of points, or a feature (as at M.A 2.2.1 in Inventor).
- ii. Positioning of indicators at the identified locations.

An important consideration regarding (i.), is that the weld indication need to be defined a way from which a trajectory can ultimately be derived: all the information necessary to define a trajectory must be included. For instance, it is not sufficient to indicate a weld path by modeling start and end points. In the 3D CAD model this would be enough information to deduce a weld path, but when stored in code, the coordinates of those points would not be sufficient. Information on which start-point belongs to which end-point would be necessary as well. The weld path information needs to be accessible and editable – it must be possible to extract and translate irrespective of the form or language that is required for further processing. It must be stored a way in which it can be extracted and exported for definition of trajectory in OLP. The weld path should be indicated in a way that allows adaption to rules added by a user.

Two alternatives for indicating weld paths have been considered. One option is to use sketches along weld lines. This has the advantage of complying with the system produced by Hagen for extracting and exporting sketches from an NX Siemens model to a post-processor (which he began to develop). It also complies with use of the predefined NX Open class for drawing sketches where bodies intersect. This alternative was ultimately pursued. A second option that may still hold value for future work was also explored: to use sketches drawn in the cross section planes of a weld. The sketches would be repeated in incrementally spaced planes along the weld lines. With this method the CAD model could be exported in a standard file type or even to a .dwg file which could be

directly imported to Visual Components. Visual Components is able to generate a robot path that moves in a straight line from one sketch to the next. Incremental spacing permits definition of a curved trajectory. This method removes the need for a post processor such as the one Hagen began to develop. Experiments were performed exporting different file types from NX with sketches drawn manually as described. The difficulty of programmatically inserting sketches in chosen planes led to ultimately pursuing the first approach.

The second highlighted challenge (ii.) is navigating a 3D model via code. One alternative is to, in the first main task, identify and extract coordinates for weld locations. This would require knowledge of how the coordinate system in any assembly model translates to code, and the ability to navigate the 3D space proficiently. Instead of using coordinates, this work opted for a looping through every component in an assembly rather than trying to navigate the 3D model. This method results in a jointed solution for the first and second main tasks: a predefined class from NX Open takes two solid bodies as input and yields output as a sketch of their intersection. This is explained under coding subtasks 4.2.3 in the next section.

4.1.3 Export weld path

Granted that the welds are indicated by sketches, as was done by Hagen, exporting the weld path can be done with the system he created. The value added here is designing a system that can draw sketches automatically. Hagen's system exported the weld path by means of a json file. Figure 5 is a flowchart of Hagen's solution adapted from his thesis. The bold arrow has been added to show where this work is applied – at the transition from a parametric model to parametric weld paths. The flowchart in Figure 9 in the next section outlines the addition to Hagen's solution provided by this work.

4.2 Code

4.2.1 Software- and language-specific system flowchart

The flowchart in Figure 9 resembles the flowchart in the methodology chapter 7. The flowchart presented in this section includes the specific software and language options that have been used in this work. As mentioned in the previous section, before a user interface is developed, the user will have to write code to edit the system default, as is the case for this system at this stage. Algorithm 1 in the next section includes a comment line indicating where a user is to add their unique conditions to the system. NX Siemens has a pre-defined class for drawing sketches at intersections between bodies or components in an assembly model. This tool can be used both interactively and programmatically. This class is applied as process (3.) in the flowchart. Process (4.) describes the storing of the weld path sketches in a dedicated variable – *weldPathSketches*. This can also be understood from the pseudocode in algorithm 1. From this step onward, the system developed by Hagen can be utilized, as it can automatically create robot code beginning with exporting sketches from an active NX Siemens session. The next section describes processes (2.) to (5.) in pseudocode.

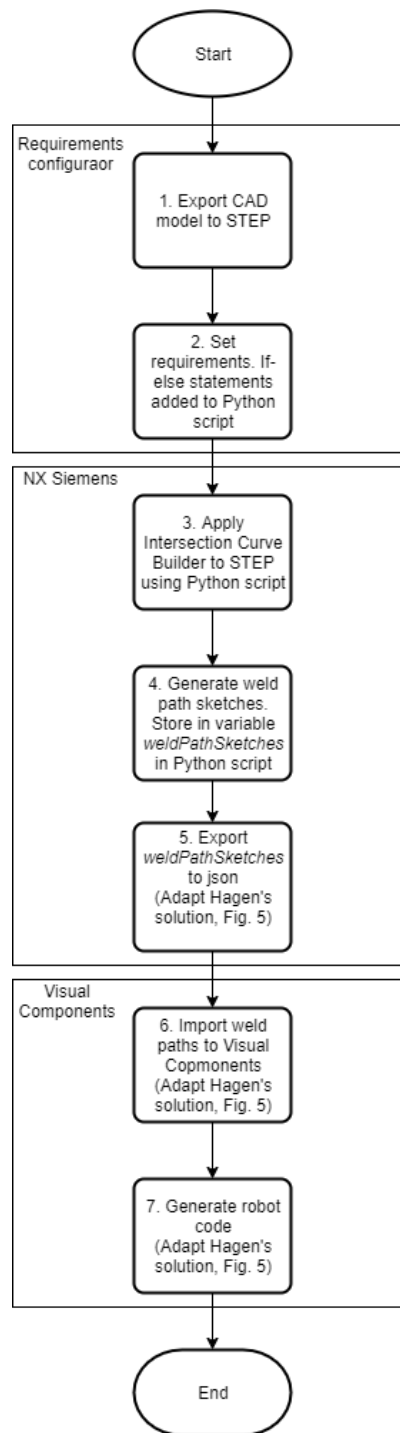


Figure 9: Software- and language-specific system flowchart

4.2.2 Pseudocode

The pseudocode below titled Algorithm 1 structures system tasks. The lines of pseudocode are elaborated on here:

- 1: First procedure starts here
- 2: The code makes a list of all the components in a STEP-file and stores them in the list *Comp*
- 3: An empty list, *weldPathSketches*, is defined for storing the sketches made by the for-loop
- 4: Index *i* loops from zero to the number of components in the list *Comp*
- 5: Index *j* loops from 1 to the number of components in the list *Comp*
- 6: Attempts the next block of code for drawing intersection sketches. If *Comp(i)* and *Comp(j)* are not in contact, the function tries the next combination of components
- 7: Draws a sketch between components in contact using the pre-defined NX class *Intersection-CurveBuilder* in Python, stores sketch in list *weldPathSketches*
- 8: Second procedure starts here
- 9: Exports the sketches defining weld path, *weldPathSketches*, to desired format at desired location
- 10: Creates an output file with the information needed for generating robot code

Algorithm 1 System pseudocode

- 1: **procedure** DRAWING SKETCHES AT INTERSECTIONS
 - 2: *Comp* ← AssemblyComponents ▷ Stores assembly components in the list *Comp*
 - 3: *weldPathSketches* ← 0 ▷ Empty list for storing welds
 - 4: **for** i=0 to length(*Comp*) **do**:
 - 5: **for** j=1 to length(*Comp*) **do**:
 - 6: **function** TRY ▷ Next block performed if *Comp(i)* and *Comp(j)* are in contact
 - ▷ Add user if-then-else conditions here
 - 7: *weldPathSketches*(i) = IntersectionCurveBuilder(*Comp*(i),*Comp*(j)) ▷ Creates sketch, stores in *weldPathSketches* list
 - 8: **procedure** EDIT/EXPORT WELDPATHSKETCHES
 - ▷ *weldPathSketches* now has list of sketches defining weld locations
 - ▷ *weldPathSketches* can be rearranged or edited here
 - 9: **function** EXPORT(*weldPathSketches*) ▷ Exports *weldPathSketches* to desired format
 - 10: **return** Output file with *weldPathSketches*
-

4.2.3 Coding subtasks

Below is a checklist of subtasks, labelled a.-j., in code integral to the development of the finished system. Progress is documented in the results chapter.

Main task 1: Find (potential) weld locations

- a. Interact with STEP-file
- b. Separate elements in STEP-file assembly
- c. Loop through model components
- d. Locate intersection lines between components
- e. Include (d.) in loop

Main task 2: Mark and store weld path

- f. Mark weld path at desired location
- g. Store path in a list or similar format

Main task 3: Define and export path

- h. If-statement in loop
- i. If-statement to filter weld locations based on condition
- j. Export weld path

Final task: Combine the above for a complete system

5 Results

5.1 Subtask results

Below are coding subtasks from section 4.2.3 which were achieved. This section refers to listings in Appendix A. Only the lines of code that are specific to the subtask are included, to save space by not attaching six full scripts.

- ✓ a. Interact with STEP-file.

With an active NX session running, a journal was recorded while opening a STEP-file. Listing A.1 shows the line of code that opens a STEP-file in an active NX session. The code can be used to open any STEP-file, by replacing the path and name in the quotation marks.

- ✓ b. Separate elements in STEP-file assembly.

STEP-files contain less information than the original file in which a model is created, e.g. model history is not included in STEP-file. In order for the proposed system to work it must be possible to recognize components in an assembly model as separate entities. By setting the selection filter to *Solid Body* or *Face* in the interactive window, it can be seen that a STEP-file assembly model does contain the information needed to separate faces and components. To confirm that separate elements could be accessed and manipulated by code, a journal was recorded while changing the color of a component, the same was done for a face. Inspection of the journal-file revealed that faces and components are numbered. By changing the number in the journal, a new face or component will change color, to a different color than used when originally recording the file. If applied, the lines of code in A.2 that are "commented out", starting with an apostrophe, programmatically add a second face that gets colored. The reason for changing colors in this and other subtasks is to visually confirm in the graphic environment that the code behaved as intended.

- ✓ c. Loop through model components.

A visual basic file for looping through bodies was found on a website dedicated to NX journaling, aptly titled <https://nxjournaling.com/>. The code titled *Creating a Subroutine to Process all Components in an Assembly* illustrates the possibility to loop through every component in an assembly [46]. The code was adapted to color components one at a time, to confirm that the components had been looped through and that the code worked. In subtask h. it was further adapted to include an if-statement, coloring based on a condition. A Python code was written to loop through every face in every component, the entire code can be found in appendix A.3. To confirm that the faces have been looped through, the code colors each face with a random color as the loop reaches it. The result can be visualized in the graphic environment and is demonstrated on two STEP-files shown in Figures 10 and 11.

- ✓ d. Locate intersection lines between components.

The Intersection Curve tool was used in the interactive environment while recording a journal file. The dialog box for the Intersection Curve tool is to the left in Figure 12. By selecting a face, all faces on the corresponding component become highlighted. The same is done with a face on an adjacent component, and a sketch is automatically drawn at their intersection. Using this tool translates to applying the *IntersectionCurveBuilder* class in the journal file. The listing for using this class can be seen in appendix A.4. The next step, for future work, is to automatically apply this class to all adjacent components in an assembly by including it in a loop.

- ✓ f. Mark weld path at desired location.

The same tool, *IntersectionCurveBuilder*, that finds intersection curves, also marks them with a sketch. The result from subtask (d.) simultaneously solves this subtask. However, in order to mark automatically all desired locations, the tool must be applied in the loop of components. As illustrated in the pseudocode Algorithm 1 in the previous section.

- ✓ h. If-statement in loop.

An if-statement was successfully added to the component loop from the NX journaling website [47]. The code from the website was edited, to color components based on their names. An example is shown in the listing in appendix A.5 which colors a component green (color = 36) if the name of the component is "nut" or "frame", Figure 13 shows the result of running the code on an assembly that contains parts with these names. The assembly was downloaded from <https://grabcad.com/> to use for testing [48]. The if-statement is necessary to have a means of filtering weld path locations, to add conditional filters to the system. The poster contribution in appendix B, Figure 15, describes this subtask and its implications.

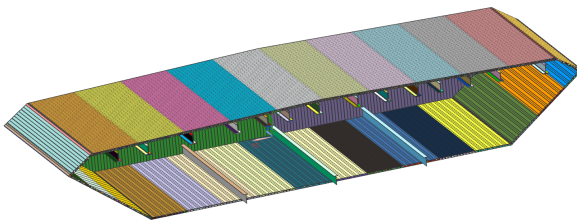


Figure 10: Subtask c. Loop through model components (demonstrated on bridge element)

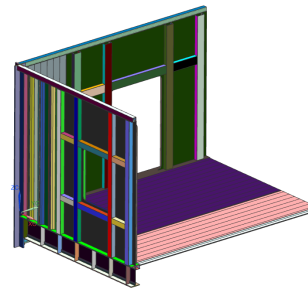


Figure 11: Subtask c. Loop through model components (demonstrated on living quarters corner)

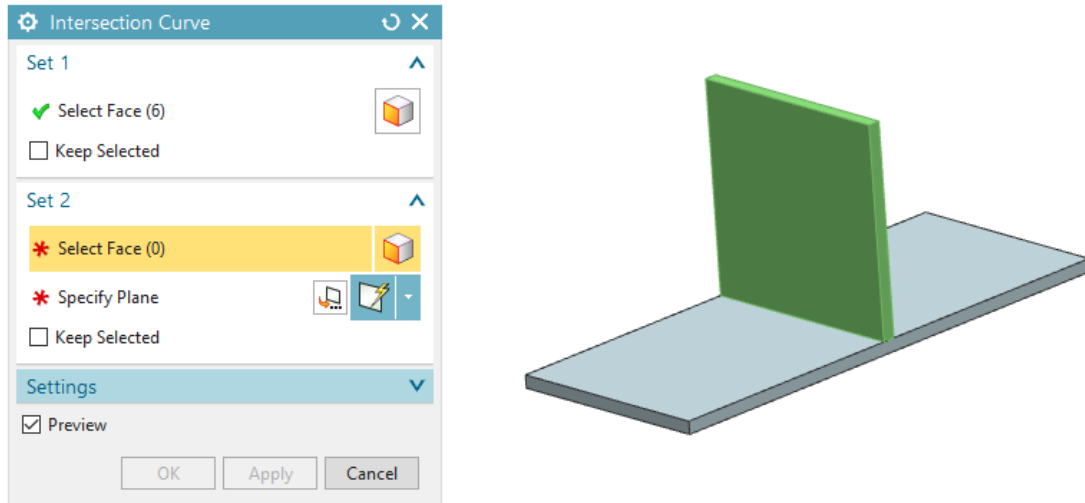


Figure 12: Subtask d. Locate intersection lines between components

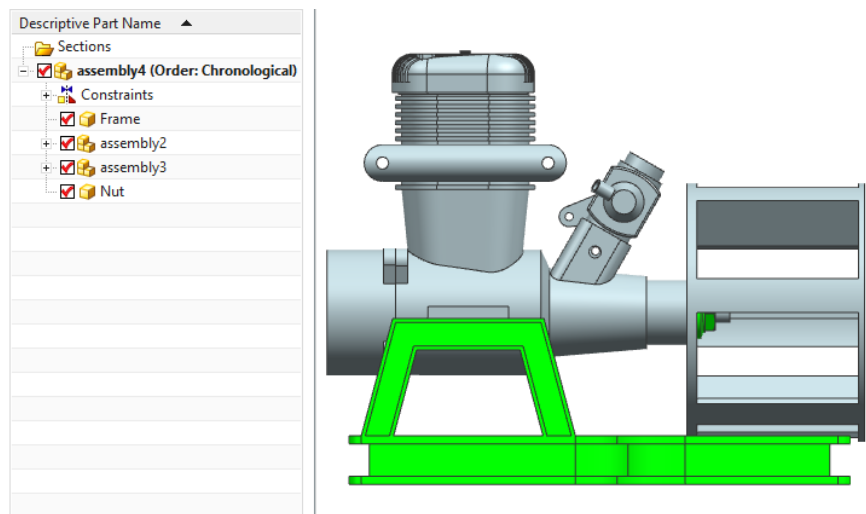


Figure 13: Subtask h. If-statement in loop (demonstrated on engine blower)

5.2 Proposal for tests of completed system

To conclude this Chapter a proposal for a test of a complete system is provided. The intention is for the test to be performed with a variety of STEP-files as input.

Check that the system...:

- ... places sketches defining weld paths and stores these in *weldPathSketches* variable by printing content of *weldPathSketches* and inspecting its content.
- ... finds the appropriate weld locations by comparing the automatically placed weld sketches with weld locations as indicated by a feature in M.A. models and in drawings in Leirvik documents.
- ... exports *weldPathSketches* by checking that the output file is generated and inspecting its content.
- ... generates robot code in Visual Components based on *weldPathSketches* by simulating trajectory in the OLP software.
- ... generates robot code from visual components by running the code, without weld gun, and checking that the motion of the robot is as intended.

6 Discussion

6.1 Interpretations and implications of findings

The system that was designed has potential to eliminate manual weld path selection, by automation. Some tasks remain before the complete system can be tested. The results show progress towards development of the system, six of ten subtasks were completed and no major obstacles were met to indicate that the system will not work. The findings suggest that the tools needed to develop the system are available. It remains to put these tools to use and complete the development. Potential improvements and modifications to the system design are considered under future work [7.1](#).

6.2 Answering research questions

- R.Q. 1. How do automatic and manual placement of welds compare?
Without a complete system it is impossible to compare automatic to manual definition of weld paths in practice. There is interest in the system in industry, provided that the completed system is efficient and accurate.
- R.Q. 2. Is design of welds repetitive enough to be automated?
Undoubtedly industries will continue to join parts by welding. In that sense, welding is repeated from one project to another. Weld-related automation would be useful for more than one project. It appears that there are enough similarities between processes that weld-design can be re-used. Assuming welds at intersection lines, is reasonable in most cases. There are assemblies where the assumption is sufficient to define the intended weld paths, for example a node model received from M.A. that required no additional filtering. There will be cases that deviate from this scenario. Even though the products in question are varied, there are commonalities that allow knowledge to be recycled from one product to the next. The answer to this R.Q. is yes. But the system must be extended to include additional requirements in a user-friendly way.
- R.Q. 3. Can the engineering knowledge of weld locations be expressed in logic terms in code?
Yes. In regard to defining the location for a weld in logic terms, this thesis concludes that: a weld line, with some exceptions, can be defined as the intersection line between components – a location which can and has been defined in code using NX's IntersectionCurveBuilder class. The next step is to enable a simple way for a user to add conditions to the system, e.g. not defining welds at locations internal to hollow profiles.
- R.Q. 4. What is the best way to transfer knowledge between platforms in development of a product that is to be welded by a robot?
Two options were considered for transferring weld path data from CAD to OLP software [4.1](#). Using a json file as an intermediate showed success in Hagen's work, illustrated in [Figure 5](#).

A new option was considered; exporting a CAD file with weld paths in a standard format, e.g. STEP or .dwg, in order to import that file directly into Visual Components. The latter method requires more investigation. The question remains unanswered.

- R.Q. 5. What is the best way to mark a weld path in CAD, with respect to transfer to software for robot code generation?

Sketches along the weld lines have been the go-to for this and for past work in this Knowledge-building project. It is worth conducting a separate study to investigate this question further, testing out all ways a weld path can be defined; sketch, feature, point, list of coordinated, etc.

- R.Q. 6. Which software tools are available for solving the present research problem, and what are their capabilities?

Most CAD allows for interaction through an API, allowing for a basic system as outlined in the methodology chapter 3. A user interface must be developed for the system for it to be efficient and accurate. Tools like ontologies can be incorporated for a system that extends beyond one platform. AI is also suggested for future work, as a means of having the system learn and improve over time.

- R.Q. 7. How can programmatic CAD be made easier?

NX Siemens has journaling capabilities for recording interactive actions to code. Some coding skills are required for editing the journal recordings, but journaling is nonetheless a useful tool for non-programmers and has been helpful in developing the system presented herein. For the system itself to be used without extensive knowledge of programmatic CAD, a user interface should be developed that allows adding requirements.

6.3 Strengths and weaknesses

1. **Navigation of CAD model through code.** Defining weld paths at component intersections, avoids having to navigate a 3D model and find the correlation between coordinate systems with code. Intersection curves are a good basis for defining welds, to which conditions can be applied to add and remove welds, adjusting the default. Progress was made toward automatically drawing intersection sketches. NX's *IntersectionCurveBuilder* automatically draws a sketch at intersections and was accessed programmatically. A code was made to loop through all components. By incorporating the *IntersectionCurveBuilder* in the loop intersection curves can be drawn automatically: this is the next step in developing the proposed system.
2. **Time investments.** The time investment required to create a KBE system and to capture knowledge in it, have been stressed by research. A KBE system was designed and the process of development begun, but not completed, during the course of this work. In a sense, high time investment has proved to be a factor in this case as well. KBE has to be evaluated in terms of rewards and costs in resources (including time). Because of the range of products for which it is applicable, an efficient and accurate system for automatic weld definition for production of large aluminum structures would most likely be worth the investment of development. This assessment is in keeping with input from industry. Leirvik has explicitly expressed the intention of obtaining a system for weld definition in 3D models.

3. **Platform independence – STEP-files and NX.** The system outlined in this thesis is designed to have a STEP-file as input with the intention of extended usage. However, to run the system an NX session has to be up and running, which means the user has to be able to connect to NX and have a license for the specific software. Ontologies are suggested for future work in developing an interconnected system. In situations where more than one vendor is being used for the same project, or in the case that a company wishes to switch vendors – a robust system would be able to tackle these transitions in a timely manner.
4. **Levels of automation.** Automation can be achieved to varying degree. On one level, automation would be achieved by the proposed system – defining and exporting weld trajectories automatically. Simultaneously, a step is made towards automation on another level – full automation. A consequence of the programmatic approach is that the same system that is being used to define welds in CAD, can be used to define and export a trajectory for OLP. In other words, the proposed system would achieve automation in its own right, while at the same time contributing towards automation on another level. Even without a fully automatic production line and with human welders only, the proposed tool would be useful.
5. **Programming.** In KBE research concerns are often raised for "black box" systems. There is a risk of having a faulty KBE system that is trusted blindly and not fully understood by users. The design and development of this system has been conducted based on a stronger foundation in CAD than in programming. Software engineering expertise is not necessary to comprehend the presented methodology. The approach of using NX's journaling capabilities in one sense automates some of the programming for the system. However, the journal output files are highly detailed, every motion of rotating the model is recorded in code, and do not represent optimal code for user interaction.
6. **User interface for adding knowledge to the system.** Vital to the success of the proposed system is the ease of adding conditions to the default weld path built from intersection curves. A user interface has not been developed, nor designed. An illustration of a user interface that can be incorporated is provided in Figure 14, the 3D model in the figure is adapted from a Leirvik test piece.

In the system presented by this thesis a user would have to edit the code defining the system itself in order to add a filter to the default settings. This is not a long-term solution. If it takes longer to input conditions to the system, than marking a weld path manually, then the system will likely not be used. Use of the intended user interface will not require substantial programming skills, perhaps none. One option is a dialog box with prompts to upload a STEP-file, options for filtering – based on for example weld length and location – and scroll-down menus for applying welding standards, as illustrated by Figure 14. Developing this system will require some programming skills, but using it will not. Yet a point for future work is for the system to learn from user input and improve its suggestions for weld path locations 7.1.
7. **Programmatic CAD.** Industries that do not use robotic welding, have not had the need to define welds in a way that can translate to robots. By defining the weld path in code, as opposed to manually or on a 2D drawing, and storing it in a variable:

- The weld path variable can be used as input for trajectory optimization
- Export to a suitable form for generating a trajectory in OLP is enabled, benefiting the goal of full automation
- The weld path can be filtered based on engineers' input to the system e.g. if the weld is shorter than x mm, do not include it.

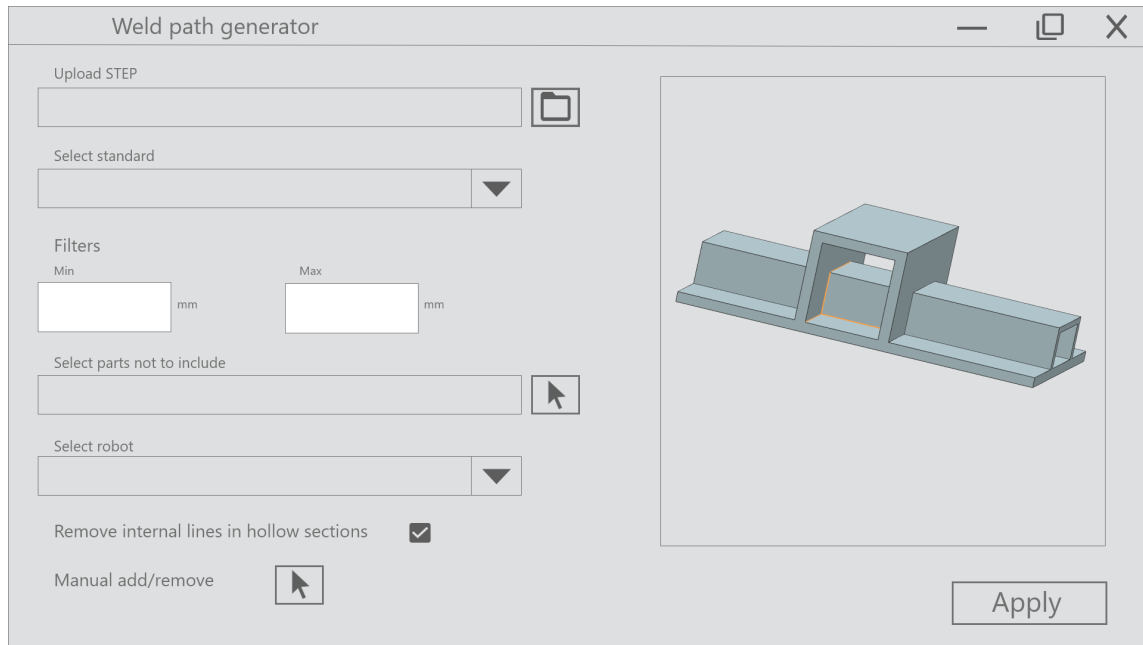


Figure 14: User interface design

7 Conclusion

The problem statement in this work has been the automatic definition of a weld path from a CAD assembly in a STEP-file. A KBE system was designed to achieve this and a number of steps were demonstrated with selected tools and technologies, further development is required to bring it to a commercially available status. The system was designed to capture knowledge of where to place welds in a given CAD model of a large assembly. It is designed to automatically export sketches drawn at intersections between components in NX Siemens. The sketches define a weld path and are the basis for a trajectory to be automatically generated in OLP. Progress was made towards development of the system. According to industry needs it is worth while to develop, granted that knowledge can be added to the system efficiently and welds placed accurately. Whether a system developed with the approach suggested in this work meets those requirements remains to be seen, it must first be developed fully and tested. Tests for a complete system are designed and presented in Results section 5.2. The next section lists future work derived from this work.

7.1 Future work

Research topics for future work based on this one are many and widespread. The list below gives suggestions for some directions which may be pursued.

1. Complete the check points from 4.2.3 and combine for a complete system
2. Perform tests from 5.2
3. Modify system to make it independent of user access to NX Siemens
4. Create user interface and develop system to easily filter where welds are defined
5. Incorporate machine learning to recognize patterns for identifying weld locations
6. Incorporate analysis in system, optimize weld locations
7. Incorporate requirements from standards in system
8. Explore ways of extending system to include Industrie 4.0 solutions
9. Extend system to produce documentation of adherence to standards, bill of materials, costs, timelines etc
10. Incorporate corrections based on real-time displacement and deformation feedback from sensors
11. Explore whether the system be modified and adopted beyond weld path definition

Bibliography

- [1] 2B1st Consulting. 2018. Rami 4.0. <https://www.2b1stconsulting.com/rami-4-0-definition/>. Accessed: 2020-16-01.
- [2] Hagen, E. B. 2018. Robot code generation from parametric models – a case study using siemens nx and visual components. Engineering and ICT masters thesis at NTNU.
- [3] The Research Council of Norway. 2020. Maroff – innovation programme for maritime activities and offshore operations. <https://www.forskningsradet.no/en/about-the-research-council/programmes/maroff---maritim-virkksomhet-og-offshore-operasjoner/>. Accessed: 2020-13-01.
- [4] Gembarski, P. C., Li, H., & Lachmayer, R. 2017. Kbe-modeling techniques in standard cad-systems: Case study—autodesk inventor professional. In *Managing Complexity*, 215–233. Springer.
- [5] Vajna, S., Weber, C., Bley, H., & Zeman, K. 2009. *CAX für Ingenieure: eine praxisbezogene Einführung*. Springer-Verlag.
- [6] SolidWorks. 2019. Solidworks api help. <https://help.solidworks.com/2019/English/api/sldworksapiproguide/Welcome.htm>. Accessed: 2019-22-12.
- [7] Autodesk Inventor. 2018. Getting started with inventor’s api. <http://help.autodesk.com/view/INVNTOR/2018/ENU/?guid=GUID-4939ABD1-A15E-473E-9376-D8208EC029EB>. Accessed: 2019-22-12.
- [8] Siemens Product Lifecycle Management Software Inc. 2014. Nx programming and customization. https://www.plm.automation.siemens.com/ru_ru/Images/4988_tcm802-4564.pdf. Accessed: 2019-16-10.
- [9] Siemens Product Lifecycle Management Software Inc. 2018. Nx siemens documentation. https://docs.plm.automation.siemens.com/tdoc/nx/12.0.2/nx_help/#uid:index. Accessed: 2020-03-01.
- [10] Frohm, J., Lindström, V., Stahre, J., & Winroth, M. 2008. Levels of automation in manufacturing. *Ergonomia-an International journal of ergonomics and human factors*, 30(3).
- [11] Ristad, M. 2019. Investigation of parametric modeling and analysis using nx and excel. Mechanical Engineering pre-masters at NTNU.

- [12] Engineering Intent. 2019. Engineering automation tools. <https://www.engineeringintent.com/engineering-automation-tools>. Accessed: 2019-30-12.
- [13] Verhagen, W. J., Bermell-Garcia, P., van Dijk, R. E., & Curran, R. 2012. A critical review of knowledge-based engineering: An identification of research challenges. *Advanced Engineering Informatics*, 26(1), 5–15.
- [14] Reijnders, A. 2012. Integrating knowledge management and knowledge-based engineering: Formal and platform independent representation of engineering rules.
- [15] Wang, L.-y., Huang, H.-h., & West, R. W. 2012. Impeller modeling and analysis based on ug nx/kf and fluent. *Journal of Central South University*, 19(12), 3430–3434.
- [16] Chapman, C. B. & Pinfold, M. 2001. The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure. *Advances in engineering software*, 32(12), 903–912.
- [17] Siemens Product Lifecycle Management Software Inc. 2017. Introduction to kbe. https://docs.plm.automation.siemens.com/tdoc/nx/12/nx_api#uid:index_fusion:id1395376:intro_kbe_ov. Accessed: 2020-04-01.
- [18] Dinh, H. T. 2015. Improving product design phase for engineer to order (eto) product with knowledge base engineering (kbe).
- [19] Inchley, G. 2014. Nx open and knowledge fusion. <https://www.eng-tips.com/viewthread.cfm?qid=371356>. Accessed: 2020-03-01.
- [20] Anderson, T. 2018. Knowledge fusion. <https://community.sw.siemens.com/s/question/0D540000061x8E1SAI/knowledge-fusion>. Accessed: 2020-03-01.
- [21] Stokes, M. et al. 2001. *Managing engineering knowledge: MOKA: methodology for knowledge based engineering applications*, volume 3. Professional Engineering Publishing London.
- [22] Sandberg, M. 2003. *Knowledge based engineering: In product development*. Luleå tekniska universitet.
- [23] Golkar, M. 2006. Development of knowledge based engineering support for design and analysis of car components using nx-knowledge fusion.
- [24] Sanya, I. & Shehab, E. 2014. An ontology framework for developing platform-independent knowledge-based engineering systems in the aerospace industry. *International Journal of Production Research*, 52(20), 6192–6215.
- [25] Cho, J., Vosgien, T., & Gerhard, D. 2017. Engineering knowledge extraction for semantic interoperability between cad, kbe and plm systems. In *IFIP International Conference on Product Lifecycle Management*, 568–579. Springer.

- [26] Lin, H.-K. & Harding, J. A. 2007. A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. *Computers in Industry*, 58(5), 428–437.
- [27] W3C. 2009. Owl web ontology language use cases and requirements. <https://www.w3.org/TR/webont-req/#onto-def>. Accessed: 2020-06-01.
- [28] Yujun, L. & Chunqing, W. 2008. Configuration knowledge expression of eto product for mass customization. In *2008 Seventh International Conference on Grid and Cooperative Computing*, 733–738. IEEE.
- [29] Technodoft Inc. 2020. Adaptive modeling language. <https://www.technosoft.com/application-software/adaptive-modeling-language/>. Accessed: 2020-06-01.
- [30] Stevenson, M., Bhungalia, A., & Zweber, J. 2000. Integrated trajectory analysis for transatmospheric vehicle design. In *8th Symposium on Multidisciplinary Analysis and Optimization*, 4817.
- [31] Drath, R. & Horch, A. 2014. Industrie 4.0: Hit or hype?[industry forum]. *IEEE industrial electronics magazine*, 8(2), 56–58.
- [32] Zezulka, F., Marcon, P., Vesely, I., & Sajdl, O. 2016. Industry 4.0—an introduction in the phenomenon. *IFAC-PapersOnLine*, 49(25), 8–12.
- [33] Lorenz, M. Industry 4.0: how intelligent machines will transform everything we know. URL: https://www.ted.com/talks/markus_lorenz_industry_4_0_how_intelligent_machines_will_transform_everything_we_know.
- [34] International Electrotechnical Commission. 2020. International standards and conformity assessment for all electrical, electronic and related technologies. <https://www.iec.ch/index.htm>. Accessed: 2020-05-01.
- [35] Willner, O., Powell, D., Gerschberger, M., & Schönsleben, P. 2016. Exploring the archetypes of engineer-to-order: an empirical analysis. *International Journal of Operations & Production Management*, 36(3), 242–264.
- [36] Agouridas, V., Allen, M., McKay, A., De Pennington, A., & Holland, S. 2001. Using information structures to gain competitive advantage. In *PICMET'01. Portland International Conference on Management of Engineering and Technology. Proceedings Vol. 1: Book of Summaries (IEEE Cat. No. 01CH37199)*, 681–692. IEEE.
- [37] Willner, O., Gosling, J., & Schönsleben, P. 2016. Establishing a maturity model for design automation in sales-delivery processes of eto products. *Computers in Industry*, 82, 57–68.
- [38] Egeland, O. Jan 2020. Smart sveis konferansen 2019.
- [39] Leirvik AS. 2020. Leirvik. <https://leirvik.com/>. Accessed: 2020-06-01.

- [40] Marine ALuminium AS. 2020. Marine aluminium. <https://m-a.no/>. Accessed: 2020-06-01.
- [41] Praveen, P. & Yarlagadda, P. 2005. Meeting challenges in welding of aluminum alloys through pulse gas metal arc welding. *Journal of Materials Processing Technology*, 164, 1106–1112.
- [42] Agapakis, J. E., Katz, J. M., Friedman, J. M., & Epstein, G. N. 1990. Vision-aided robotic welding: an approach and a flexible implementation. *The International Journal of Robotics Research*, 9(5), 17–34.
- [43] Tarn, T.-J., Chen, S.-B., & Zhou, C. 2007. *Robotic welding, intelligence and automation*, volume 362. Springer.
- [44] Wittenberg, G. 1995. Developments in offline programming: an overview. *Industrial Robot: An International Journal*, 22(3), 21–23.
- [45] Maw, I. 2019. The what, why and how of industrial robot simulation software for offline programming (olp). <https://www.engineering.com/AdvancedManufacturing/ArticleID/18288/>. Accessed: 2020-05-01.
- [46] NX Journaling. 2012. Creating a subroutine to process all components in an assembly. <https://nxjournaling.com/content/creating-subroutine-process-all-components-assembly>. Accessed: 2020-13-01.
- [47] NX Journaling. 2020. Engineer meet nx journaling. <https://nxjournaling.com/>. Accessed: 2020-13-01.
- [48] Kumar, N. 2018. Engine blower. <https://grabcad.com/>. Accessed: 2020-13-01.

A Code and testing

A.1 Program listings for subtasks

This section includes listings from the Results chapter 5. Note: Some subtasks were performed in Visual Basic, the default journaling language in NX.

Code Listing A.1: Subtask a. Interact with STEP-file

```
basePart1 , partLoadStatus1 =
theSession.Parts.OpenActiveDisplay("M:\\CAD19\\sharp_angle_part.STEP",
NXOpen.DisplayPartOption.AllowAdditional)
```

Code Listing A.2: Subtask b. Separate elements in STEP-file assembly

```
displayModification1.NewColor = 200

displayModification1.NewWidth =
NXOpen.DisplayableObject.ObjectWidth.One

Dim objects1(0) As NXOpen.DisplayableObject
'Dim objects2(0) As NXOpen.DisplayableObject

Dim brep1 As NXOpen.Features.Brep =
CType(workPart.Features.FindObject("UNPARAMETERIZED_FEATURE(1)"),
NXOpen.Features.Brep)

Dim face1 As NXOpen.Face =
CType(brep1.FindObject("FACE_2_{(35,5,-10)}_UNPARAMETERIZED_FEATURE(1)"),
NXOpen.Face)

'Dim face2 As NXOpen.Face =
CType(brep1.FindObject("FACE_9_{(35,5,-10)}_UNPARAMETERIZED_FEATURE(1)"),
NXOpen.Face)

objects1(0) = face1
'objects2(0) = face2

displayModification1.Apply(objects1)
'displayModification1.Apply(objects2)
```

Code Listing A.3: Subtask c. Loop through model components

```

import random
import NXOpen

def main():
    #Gets the current session and work parts from session
    theSession = NXOpen.Session.GetSession()
    workPart = theSession.Parts.Work

    #object for showing dialogue boxes
    theNxMessageBox = NXOpen.UI.GetUI().NXMessageBox

    #collect all objects on a particular layer (default is 1)
    objects = workPart.Layers.GetAllObjectsOnLayer(1)

    #Loops through all objects
    for object in objects:
        try:
            #attempts to check if object is a body
            if object.IsSolidBody:

                #if object is a body, collect all faces
                faces = object.GetFaces()

                #loops through faces
                for face in faces:

                    #random color
                    randomColor = random.randint(1,216)
                    face.Color = randomColor
                    #refresh display of face to show color
                    face.RedisplayObject()

            #if error is met, do nothing
        except Exception as e:
            pass

if __name__ == '__main__':
    main()

```

Code Listing A.4: Subtask d. Locate intersection lines between components

```

# NX 12.0.2.9
# Journal created by sapresco on Sat Jan 11 09:46:45 2020 W. Europe Standard Time
#
import math
import NXOpen
import NXOpen.Assemblies

```

```

import NXOpen.Features
import NXOpen.GeometricUtilities
def main() :

theSession = NXOpen.Session.GetSession()
workPart = theSession.Parts.Work
displayPart = theSession.Parts.Display
# -----
#   Menu: Insert->Derived Curve->Intersect...
# -----
markId1 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Visible, "Start")

intersectionCurveBuilder1 =
workPart.Features.CreateIntersectionCurveBuilder(NXOpen.Features.Feature.Null)

origin1 = NXOpen.Point3d(0.0, 0.0, 0.0)
normal1 = NXOpen.Vector3d(0.0, 0.0, 1.0)
plane1 =
workPart.Planes.CreatePlane(origin1, normal1,
NXOpen.SmartObject.UpdateOption.WithinModeling)

unit1 = workPart.UnitCollection.FindObject("MilliMeter")
expression1 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

expression2 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

origin2 = NXOpen.Point3d(0.0, 0.0, 0.0)
normal2 = NXOpen.Vector3d(0.0, 0.0, 1.0)
plane2 =
workPart.Planes.CreatePlane(origin2, normal2,
NXOpen.SmartObject.UpdateOption.WithinModeling)

expression3 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)
expression4 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

intersectionCurveBuilder1.CurveFitData.Tolerance = 0.01

intersectionCurveBuilder1.CurveFitData.AngleTolerance = 0.5

theSession.SetUndoMarkName(markId1, "Intersection_Curve_Dialog")

part1 = theSession.Parts.FindObject("P2_id31_stp")
partLoadStatus1 = part1.LoadThisPartFully()

partLoadStatus1.Dispose()
extractFaceBuilder1 =
workPart.Features.CreateExtractFaceBuilder(NXOpen.Features.Feature.Null)

extractFaceBuilder1.Type = NXOpen.Features.ExtractFaceBuilder.ExtractType.Face

```



```
extractFaceBuilder1.FaceOption =
NXOpen.Features.ExtractFaceBuilder.FaceOptionType.FaceChain

extractFaceBuilder1.ParentPart =
NXOpen.Features.ExtractFaceBuilder.ParentPartType.OtherPart

extractFaceBuilder1.Associative = True

scCollector1 = extractFaceBuilder1.FaceChain

component1 =
workPart.ComponentAssembly.RootComponent.FindObject("COMPONENT_P2_id31_stp1")
body1 = component1.FindObject("PROTO#.Bodies|P2(1)")
faceBodyRule1 = workPart.ScRuleFactory.CreateRuleFaceBody(body1)

rules1 = [None] * 1
rules1[0] = faceBodyRule1
scCollector1.ReplaceRules(rules1, False)

extractFaceBuilder1.FixAtCurrentTimestamp = True

markId2 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible, None)

feature1 = extractFaceBuilder1.CommitCreateOnTheFly()

theSession.DeleteUndoMark(markId2, None)

waveLinkRepository1 = workPart.CreateWavelinkRepository()

waveLinkRepository1.SetNonFeatureApplication(False)

waveLinkRepository1.SetBuilder(intersectionCurveBuilder1)

extractFace1 = feature1
waveLinkRepository1.SetLink(extractFace1)

extractFaceBuilder2 = workPart.Features.CreateExtractFaceBuilder(extractFace1)

extractFaceBuilder2.Associative = False

markId3 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible, None)

feature2 = extractFaceBuilder2.CommitCreateOnTheFly()

extractFaceBuilder2.Destroy()

theSession.DeleteUndoMark(markId3, None)

extractFaceBuilder1.Destroy()
```

```

features1 = [NXOpen.Features.Feature.Null] * 1
extractFace2 = feature2
features1[0] = extractFace2
faceFeatureRule1 = workPart.ScRuleFactory.CreateRuleFaceFeature(features1)

rules2 = [None] * 1
rules2[0] = faceFeatureRule1
intersectionCurveBuilder1.FirstFace.ReplaceRules(rules2, False)

features2 = [NXOpen.Features.Feature.Null] * 1
features2[0] = extractFace2
faceFeatureRule2 = workPart.ScRuleFactory.CreateRuleFaceFeature(features2)

rules3 = [None] * 1
rules3[0] = faceFeatureRule2
intersectionCurveBuilder1.FirstFace.ReplaceRules(rules3, False)

objects1 = [NXOpen.TaggedObject.Null] * 6
face1 =
extractFace2.FindObject("FACE_1_{(96.2358551025391,16.5,18.5)_LINKED_FACE(0)}")
objects1[0] = face1
face2 =
extractFace2.FindObject("FACE_2_{(98.2358551025391,16.5,18.5)_LINKED_FACE(0)}")
objects1[1] = face2
face3 =
extractFace2.FindObject("FACE_3_{(97.2358551025391,16.5,35)_LINKED_FACE(0)}")
objects1[2] = face3
face4 =
extractFace2.FindObject("FACE_4_{(97.2358551025391,33,18.5)_LINKED_FACE(0)}")
objects1[3] = face4
face5 =
extractFace2.FindObject("FACE_5_{(97.2358551025391,16.5,2)_LINKED_FACE(0)}")
objects1[4] = face5
face6 =
extractFace2.FindObject("FACE_6_{(97.2358551025391,0,18.5)_LINKED_FACE(0)}")
objects1[5] = face6
added1 = intersectionCurveBuilder1.FirstSet.Add(objects1)

part2 = theSession.Parts.FindObject("P1_id68_stp")
partLoadStatus2 = part2.LoadThisPartFully()

partLoadStatus2.Dispose()
extractFaceBuilder3 =
workPart.Features.CreateExtractFaceBuilder(NXOpen.Features.Feature.Null)

extractFaceBuilder3.Type = NXOpen.Features.ExtractFaceBuilder.ExtractType.Face

extractFaceBuilder3.FaceOption =
NXOpen.Features.ExtractFaceBuilder.FaceOptionType.FaceChain

```

```
extractFaceBuilder3.ParentPart =
NXOpen.Features.ExtractFaceBuilder.ParentPartType.OtherPart

extractFaceBuilder3.Associative = True

scCollector2 = extractFaceBuilder3.FaceChain

component2 =
workPart.ComponentAssembly.RootComponent.FindObject("COMPONENT_P1_id68_stp_1")
body2 = component2.FindObject("PROTO#.Bodies|P1(1)")
faceBodyRule2 = workPart.ScRuleFactory.CreateRuleFaceBody(body2)

rules4 = [None] * 1
rules4[0] = faceBodyRule2
scCollector2.ReplaceRules(rules4, False)

extractFaceBuilder3.FixAtCurrentTimestamp = True

markId4 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible, None)

feature3 = extractFaceBuilder3.CommitCreateOnTheFly()

theSession.DeleteUndoMark(markId4, None)

extractFace3 = feature3
waveLinkRepository1.SetLink(extractFace3)

extractFaceBuilder4 = workPart.Features.CreateExtractFaceBuilder(extractFace3)

extractFaceBuilder4.Associative = False

markId5 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible, None)

feature4 = extractFaceBuilder4.CommitCreateOnTheFly()

extractFaceBuilder4.Destroy()

theSession.DeleteUndoMark(markId5, None)

extractFaceBuilder3.Destroy()

features3 = [NXOpen.Features.Feature.Null] * 1
extractFace4 = feature4
features3[0] = extractFace4
faceFeatureRule3 = workPart.ScRuleFactory.CreateRuleFaceFeature(features3)

rules5 = [None] * 1
rules5[0] = faceFeatureRule3
intersectionCurveBuilder1.SecondFace.ReplaceRules(rules5, False)
```

```

features4 = [NXOpen.Features.Feature.Null] * 1
features4[0] = extractFace4
faceFeatureRule4 = workPart.ScRuleFactory.CreateRuleFaceFeature(features4)

rules6 = [None] * 1
rules6[0] = faceFeatureRule4
intersectionCurveBuilder1.SecondFace.ReplaceRules(rules6, False)

objects2 = [NXOpen.TaggedObject.Null] * 6
face7 =
extractFace4.FindObject("FACE_1_{(97.2358551025391,16.5,0)_LINKED_FACE(1)}")
objects2[0] = face7
face8 =
extractFace4.FindObject("FACE_2_{(97.2358551025391,16.5,2)_LINKED_FACE(1)}")
objects2[1] = face8
face9 =
extractFace4.FindObject("FACE_3_{(97.2358551025391,33,1)_LINKED_FACE(1)}")
objects2[2] = face9
face10 =
extractFace4.FindObject("FACE_4_{(137.7358551025391,16.5,1)_LINKED_FACE(1)}")
objects2[3] = face10
face11 =
extractFace4.FindObject("FACE_5_{(97.2358551025391,0,1)_LINKED_FACE(1)}")
objects2[4] = face11
face12 =
extractFace4.FindObject("FACE_6_{(56.7358551025391,16.5,1)_LINKED_FACE(1)}")
objects2[5] = face12
added2 = intersectionCurveBuilder1.SecondSet.Add(objects2)

markId6 =
theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible,
"Intersection_Curve")

nXObject1 = intersectionCurveBuilder1.Commit()

theSession.DeleteUndoMark(markId6, None)

theSession.SetUndoMarkName(markId1, "Intersection_Curve")

intersectionCurveBuilder1.Destroy()

try:
    # Expression is still in use.
    workPart.Expressions.Delete(expression2)
except NXOpen.NXException as ex:
    ex.AssertErrorCode(1050029)

try:
    # Expression is still in use.

```

```
workPart.Expressions.Delete(expression4)
except NXOpen.NXException as ex:
    ex.AssertErrorCode(1050029)

try:
    # Expression is still in use.
    workPart.Expressions.Delete(expression1)
except NXOpen.NXException as ex:
    ex.AssertErrorCode(1050029)

plane1.DestroyPlane()

try:
    # Expression is still in use.
    workPart.Expressions.Delete(expression3)
except NXOpen.NXException as ex:
    ex.AssertErrorCode(1050029)

plane2.DestroyPlane()

waveLinkRepository1.Destroy()

markId7 = theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Visible, "Start")

intersectionCurveBuilder2 =
workPart.Features.CreateIntersectionCurveBuilder(NXOpen.Features.Feature.Null)

origin3 = NXOpen.Point3d(0.0, 0.0, 0.0)
normal3 = NXOpen.Vector3d(0.0, 0.0, 1.0)
plane3 =
workPart.Planes.CreatePlane(origin3, normal3,
NXOpen.SmartObject.UpdateOption.WithinModeling)

expression5 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

expression6 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

origin4 = NXOpen.Point3d(0.0, 0.0, 0.0)
normal4 = NXOpen.Vector3d(0.0, 0.0, 1.0)
plane4 =
workPart.Planes.CreatePlane(origin4, normal4,
NXOpen.SmartObject.UpdateOption.WithinModeling)

expression7 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

expression8 = workPart.Expressions.CreateSystemExpressionWithUnits("0", unit1)

intersectionCurveBuilder2.CurveFitData.Tolerance = 0.01

intersectionCurveBuilder2.CurveFitData.AngleTolerance = 0.5
```

```

theSession.SetUndoMarkName(markId7, "Intersection_Curve_Dialog")

# -----
#   Dialog Begin Intersection Curve
# -----
# -----
#   Menu: Tools->Journal->Stop Recording
# -----

if __name__ == '__main__':
    main()

```

Code Listing A.5: Subtask h. If-statement in loop

```

'SARAH EDITS START

If comps(i).displayname = "Frame" Or comps(i).displayname = "Nut" Then

Dim displayModification1 As NXOpen.DisplayModification = Nothing
displayModification1 = theSession.DisplayManager.NewDisplayModification()

displayModification1.ApplyToAllFaces = True

displayModification1.ApplyToOwningParts = False

displayModification1.NewColor = 36

Dim objects1(0) As NXOpen.DisplayableObject

objects1(0) = component1
displayModification1.Apply(objects1)

Dim nErrs1 As Integer = Nothing

displayModification1.Dispose()

End If
'SARAH EDITS END

```

B Publishability

B.1 MTP-conference poster

Poster-contribution to the first annual MTP-conference at NTNU in 2019 can be seen in Figure 15.

B.2 FAIM 2020 (1) abstract accepted, paper to be submitted by February 2020

The accepted abstract for this conference can be seen in Figure 16. The conference is described here: "*Flexible Automation and Intelligent Manufacturing International Conference series has been run in 26 cities in 14 countries in Europe, America and Asia, addressing both technology and management aspects.*"..."*The theme of FAIM2020 is 'Multiple, complementary and evolving facets of modern manufacturing: holistic synthesis' pointing to its four thematic pillars underpinned by automation and intelligence streams: Manufacturing processes, Machine tools and manufacturing equipment, Manufacturing systems, Enabling technologies.*" <https://www.faimconference.org/>

B.3 FAIM 2020 (2) paper submitted

Co-authored *Formalization of engineering knowledge for industrial robots using Knowledge Fusion language* to be submitted to FAIM 2020. First page can be seen in Figure 17.

Potential of programmatic CAD

Automation of design and production

Simplifying the programmatic use of CAD, to accommodate for Knowledge-based engineering of large welded structures

INTRO

- Knowledge-based Engineering (KBE), capture and re-use of engineering knowledge – one interpretation: through code.
- Programmatic use of CAD: store and apply model-knowledge with benefits of using programming language.
- Gap: maturity of KBE.

METHODS

For the case of large welded structures. Code developed to:

1. loop through components in CAD assembly.
2. ID weld locations and draw sketch to mark trajectory.
3. export trajectory to software for generating robot code.

RESULTS (DEMO)

- Code loops through assembly components and edits based on IF-statement.
- Example: Visual Basic code (left), run on staircase structure (right), colors component based on its name: "baseframe1" or "baseframe2".

```

For i As Integer = 0 To (comps.Length - 1)
Dim component1 As Assemblies.Component = comps(i)
Dim components1(0) As Assemblies.Component
components1(0) = component1
[...]
```

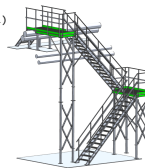
```

If comps(i).displayname = "baseframe1" Or
comps(i).displayname = "baseframe2" Then
Dim displayModification1 As
NXOpen.DisplayModification = Nothing
[...]
```

```

displayModification1.ApplyToAllFaces = True
displayModification1.ApplyToOwningParts = False
displayModification1.NewColor = 36

```



[1]

[2]

DISCUSSION AND NEXT STEPS

- Code has been made to loop through components in any assembly in NX and edit based on condition.
- NX Open APIs have classes and functions for identifying constraints and drawing sketches.
- Based on these classes weld trajectory (defined by sketch) is to be automatically created.

Part of Knowledge-building Project for Industry – KPN^[3]

- MAROFF – Innovation Programme for Maritime Activities and Offshore Operations
- **Robotic welding of large Aluminum structure**
- Partners:
 - Hydro Aluminium
 - Leirvik
 - DIGITREAD


System architecture:



Image: Morten Fossmark
– based on ICIT2020 paper^[4]

Programmatic approach – opportunities for KBE, automation and Industrie 4.0^[5]



 Sarah Prescott, Andrei Lobov, Morten Fossmark, Tuan Anh Tran
Department of Mechanical and Industrial Engineering
Norwegian University of Science Technology

 NTNU



Scan for citations and additional information



Figure 15: MTP-conference poster 2019



**30th International Conference on
Flexible Automation and Intelligent Manufacturing**

15-18 June 2020, Athens, Greece

Hosted by National Technical University of Athens, School of Mechanical Engineering

Simplifying the programmatic use of CAD software to accommodate for Knowledge-Based Engineering

Sarah Ann Oxman Prescott

Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

Abstract

Knowledge based engineering (KBE) is often described as the capture and re-use of engineering knowledge. One interpretation of this is the digital storage and application of engineering knowledge in code form. Coding skills are not required for use of CAD, which may be perceived as an advantage. It may seem counterintuitive to work with CAD programmatically rather than interactively. However, programmatic use of CAD offers major benefits related to KBE. Despite KBE's potential benefits, its practice is not an industry norm. Many attribute this to the cost in time of applying KBE practices, partially due to the difficulty of programmatic use of CAD. KBE has been under development since the 1980s. Several cases of KBE have been presented in literature. These cases often endorse KBE and boast of its potential, but neglect to address the specific challenges associated with implementation of KBE. This paper eases the programmatic use of CAD to enable KBE for the case of large welded aluminium structures. A set of python classes are curated and instructions are developed to accompany these classes. The classes with instructions explain how to programmatically identify weld lines in a given CAD model and extract a weld trajectory that can be exported and used to generate robot code for a welding robot.

Keywords

Knowledge based engineering (KBE), automation, robot welding

Figure 16: Accepted abstract for FAIM 2020

Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 00 (2019) 000–000

Procedia
MANUFACTURINGwww.elsevier.com/locate/procedia

30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2020)
15-18 June 2020, Athens, Greece.

Formalization of engineering knowledge for industrial robots using Knowledge Fusion language

Andrei Lobov, Tuan Anh Tran, Sarah Ann Oxman Prescott

Norwegian University of Science and Technology, 7034, Trondheim, Norway

* Corresponding author. Tel.: +47-45-913-455; E-mail address: andrei.lobov@ntnu.no

Abstract

Knowledge-based engineering aims to automate product design processes. Design for manufacturability (DFM) aims to create products that can be made in the simplest way, i.e. that do not demand any unnecessary resources and use the simplest possible procedures to still meet the product requirements. Robots have a key role in automation of manufacturing processes. Use of formalized representation of engineering knowledge for industrial robots' application can allow for better decision making also for DFM requirements. This paper proposes an approach for integrating product design with robot applications using Knowledge Fusion language. An approach is illustrated with Siemens NX tools framework.

© 2020 The Authors. Published by Elsevier Ltd.

This is an open access article under the CC BY-NC-ND license <https://creativecommons.org/licenses/by-nc-nd/4.0/>

Peer-review under responsibility of the scientific committee of the FAIM 2020.

Keywords: knowledge-based engineering; product design; robotics; manufacturing.

1. Introduction

Robots can support different processes at the production floor. Examples of usage are i) machine tending, e.g. loading and unloading CNC machines; ii) performing various material handling tasks, e.g. for packaging or quality inspection; iii) performing assembly operations, e.g. picking and placing components or subassemblies to the right position; and iv) supporting manufacturing processes with a specific tool, e.g. for joining operations holding a screwdriver or a welding gun.

Robot vendors develop robot programming interfaces or software suites to simplify the engineering process of a robot work cell. These suites are often built with focus on application for specific robots with little capability to develop and share outside the lessons learnt via a limited number of robotics projects.

On the product design side, a common tool for product engineers is a Computer-aided Design (CAD) software. A common approach for using such a tool is a sequential

modification of 3D bodies with basic operations on those such as Unite, Subtract, Extrude, etc., towards a desired outcome – a 3D model of a product that besides promoting a better understanding by being visualized, can also be checked for some desired properties, e.g. behavior with respect to acting forces, materials, fatigue or the application of different manufacturing processes modelled in a Computer-aided Manufacturing (CAM) package often accompanying CAD software.

Also, for CAD/CAM software packages, the engineering knowledge often remains with the human engineers, i.e. in non-formal, non-machine processible representation.

A knowledge-based engineering (KBE) paradigm that emerged in the 1980s tries to formalize the knowledge as a set of engineering rules that can be processed by a computer, for example, in parallel with a CAD system to suggest better design options. Various languages have emerged for engineering knowledge representation. Reference [1] lists the following languages used today, including GDL, KNEXT,

2351-9789 © 2020 The Authors. Published by Elsevier Ltd.

This is an open access article under the CC BY-NC-ND license <https://creativecommons.org/licenses/by-nc-nd/4.0/>

Peer-review under responsibility of the scientific committee of the FAIM 2020.

Figure 17: Formalization of engineering knowledge for industrial robots using Knowledge Fusion language

