Generating Human Work Instructions from Assembly Plans

Csaba Kardos[a,b], András Kovács[a], Balázs E. Pataki[c], József Váncza[a,b]
[a]EPIC Center of Excellence in Production Informatics and Control, Inst. Comp. Sci. & Control, Hun. Acad. Sci.
[b]Department of Manufacturing Science and Technology, Budapest University of Technology and Economics,
[c]Department of Distributed Systems, Institute for Computer Science and Control, Hungarian Academy of Sciences,
{csaba.kardos, andras.kovacs, pataki.balazs, vancza}@sztaki.mta.hu

Abstract

Despite enormous robotization efforts, most of the assembly process is still executed by human workforce in many industries performing the assembly of mechanical products. Therefore, a crucial component of any automated planning system in those applications is the worker instruction system that presents the automatically generated plans to human assembly workers. In case of complex products and processes, finding the most efficient presentation to workers with different skills and background is a great challenge. This paper proposes novel methods for generating context-dependent, animated work instructions from automatically generated assembly plans. The proposed approach is demonstrated on an industrial case study that involves the manual assembly of an automotive supercharger.

Introduction

While a significant portion of the research on automated planning and scheduling focuses on decision making in autonomous systems, in many applications, the generated plans are still executed by human actors. In such applications, a key success factor is the effective instruction system that presents the complex plan to the human worker. This is the case in mechanical assembly as well: despite intensive robotization efforts, the assembly of mechanical products in many industries still requires human workforce.

The motivation for replacing classical static, often still paper-based work instructions by enhanced digital Worker Instruction Systems (WIS) and automatically generated instructions is manifold. Firstly, shortening product life cycles and increasing variety make it difficult to maintain and regularly update static instructions, and hence, the ability to dynamically adjust the digital content to the changing production environment is the most attractive characteristic of WIS (Leu et al. 2013; Lušić et al. 2014). Most commercial WIS implementations offer interfaces to Enterprise Resource Planning (ERP), Product Lifecycle Management (PLM) and Manufacturing Execution Systems (MES), which enables the import of product and process data, supports the definition of instruction templates, and hence, allows maintaining consistent and up-to-date work instructions.

In line with the above, vast effort is being invested into *computer-aided process planning* (CAPP) techniques to manage the complexity of planning, and to support the process engineer in defining the most efficient assembly processes for the products (Hu et al. 2011; Ghandi and Masehian 2015). As the final step of the process planning workflow is the generation of program codes for robotic resources and instructions for human workers, the automation of process planning is incomplete without automated methods for instruction generation as well (Kaipa et al. 2012).

Finally, it must be recognized that the increasing complexity of products and production systems puts a heavy cognitive load on assembly workers as well (Leu et al. 2013; Lušić et al. 2016). Therefore, human workers need enhanced support from the WIS, in the form of unambiguous instructions delivered using the modality that suits the given environment the best; in addition to classical text instructions, figures, videos, 3D animations, audio instructions, or even augmented reality (AR) can be used, too. Instructions can be tailored further by exploiting context awareness (Bader and Aehnelt 2014; Bannat et al. 2008; Claeys et al. 2016). Efficient WIS and instructions have a special role in supporting the training of the workers (Michalos et al. 2014) and in solving one-of-a-kind issues in maintenance and rework operations (Fiorentino et al. 2014).

This paper presents novel methods for the automated generation of work instructions for mechanical assembly. The methods proposed in this paper support the final step of the workflow in a full-fledged workcell configuration toolbox, which includes efficient optimization methods for assembly sequence planning, resource assignment (i.e., action planning) and path planning, by presenting the resulting plans to human workers. For fully exploiting the opportunities of digital WIS, the methods support the generation of context-aware instructions augmented with 3D animations of the assembly process.

Problem Definition

The objective of *assembly workcell configuration* is to manage the journey of assembled products on their way from design ideas to the reality of production. Its two main sub-problems, which approach the same problem from the viewpoints of the assembly process and the assembly resources, respectively, are *assembly process planning* and *workcell layout design*. Both of these sub-problems involve various

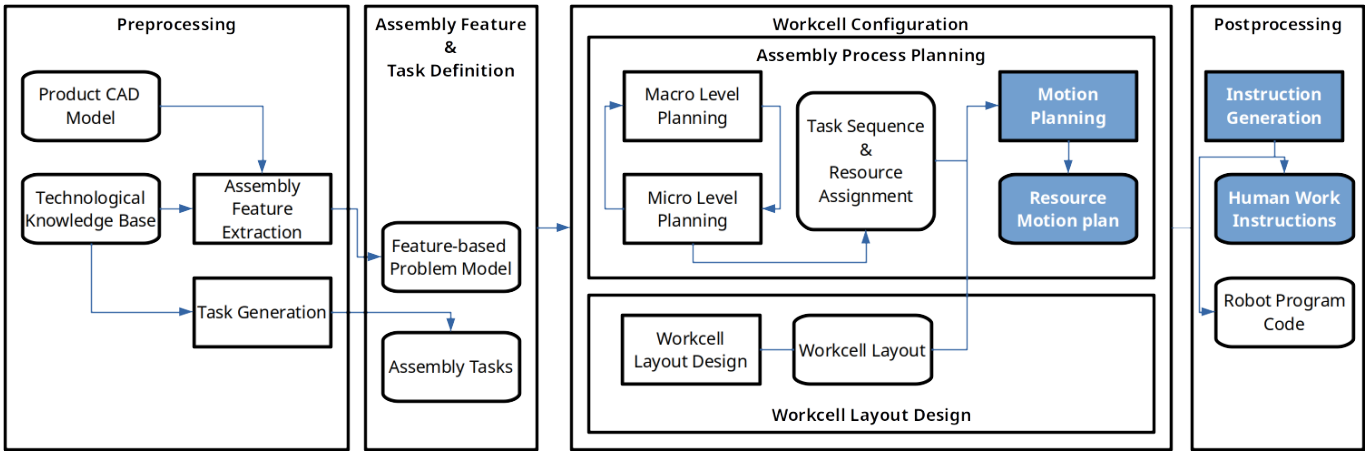


Figure 1: Workflow of assembly workcell process planning. The sub-problem in scope, instruction generation, is highlighted in blue.

types of decisions. In case of assembly process planning, it is common to differentiate *macro-level planning*, which is a combinatorial problem involving decisions on the assembly task sequence and the resources assigned to the *tasks*; and micro-level planning, which is responsible for elaborating the details of each individual task. The latter includes, e.g., assembly path planning, fixturing, or tooling. The workflow of assembly workcell configuration applied in this paper, departing from the conversion of legacy CAD models of the product into feature-based models directly processable for automated planners, and concluding at the generation of human work instructions and robot program codes, is presented in Fig. 1.

In case of assembly tasks performed by human workers, the final step of the workcell configuration workflow is the generation of *work instructions* for the workers, which is the focus of this paper. Instructions must be generated from a structured representation of the assembly process plan, which includes the sequence of assembly tasks, a feature-based representation of the technological content of each task, as well as a collision-free assembly path. It has to be noted that in an actual manufacturing environment, human supervision and interaction are still inevitable before finalizing instructions. Therefore, there are two main user roles in an instruction generation system: process engineers who participate in the creation of instructions, and workers to whom the instructions are delivered via the multi-modal interfaces. The goal here is to automatically generate draft instructions that will be verified and finalized by a process engineer. Thus, effective work instructions must fulfill the following key requirements:

- Instructions must be easily and unambiguously interpretable by the workers. For this purpose, classical textual instructions can be accompanied by images or animations.
- Context-dependent instructions should be applied to fit the actual conditions, e.g., the worker's skill, language, devices and presentation preferences, as much as possible.
- Instructions should be delivered using the modality that suits the actual application the best. Beyond instructions displayed on screens, audio instructions are also in scope.
- The generation of instruction should be automated as much as possible. Nevertheless, the system should enable editing and approving the instructions by a process engineer.
- It should be possible to generate the instructions from legacy representations, such as the existing CAD models of the products.

In the sequel, a brief overview is given of the overall methodology applied to assembly workcell configuration, focusing on sub-problems related to assembly process planning, and then the proposed approach to work instruction generation is presented in detail.

Assembly Planning Approach

Feature-based Assembly Model

A key challenge in process planning in any manufacturing domain is to match the different views of the planning problem, related to geometry, technology, tooling, etc. The classical approach to responding to this challenge is to use a so-called *feature-based* representation, in which features give a holistic description of the micro-worlds related to an individual manufacturing or assembly operation. Examples of features in assembly include *placing* two parts on each other, *insertion* of a part into a hole on another part, or *screwing*. The complete specification of such a feature then consists of the geometries of the corresponding parts, the relative position of the parts in the target configuration, and the definition of the movement that can join the parts, including the direction of the motion and the technological parameters. The model also allows organizing individual features that belong together in a sense, e.g., parallel screwing features that join the same parts, into so-called *composite features*. The assembly planner then assigns a single assembly task to the composite feature, which involves creating all in-

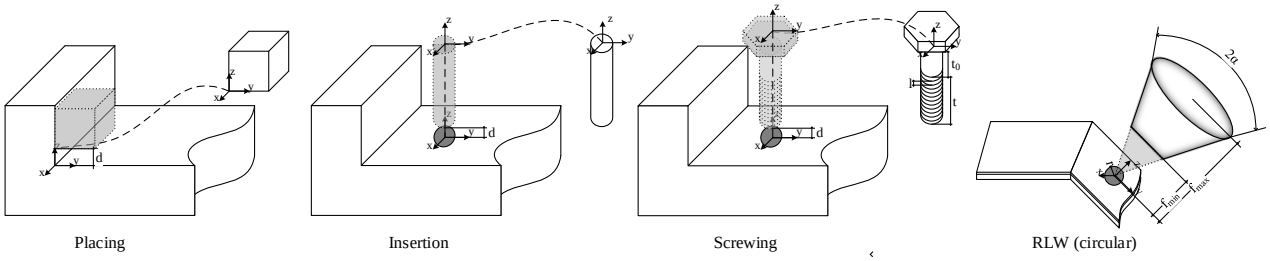


Figure 2: Types of assembly features and their parameters.

dividual features in parallel or in a given sequence, as defined by the ordering flag in the composite (see details later). Some assembly features and their parameters are presented in Fig. 2, whereas a generic presentation of feature-based assembly planning is given in (Wang, Keshavarzmanesh, and Feng 2011).

Formally, there is given a set of assembly tasks (and potentially some auxiliary tasks) that must be executed in an appropriate order using suitable resources to arrive from the set of parts into an assembled product. The technological content of each assembly task is specified by the assembly feature assigned to it, which defines the set of base parts and moved parts, along with the parameters of the motion that joins the two sets of parts (Fig. 2). This motion can be subdivided into two phases: an *approach* motion that takes the parts from their storage locations to a so-called *near* position using an arbitrary collision-free trajectory, and the *local* motion from the near position into the final configuration. The latter, local motion is perfectly defined in the assembly feature. It is assumed that each assembly task requires two resources for its execution: a fixture that holds the base parts at their place and a tool that executes the task on the moved parts. The assembly planner also adds so-called changeover tasks to the plan at points where some of the assigned resources differ between consecutive assembly tasks. The details of the feature-based assembly process planning model are presented in (Kardos, Kovács, and Váncza 2016; 2017).

Assembly Sequence Planning and Resource Assignment

Since assembly process planning involves making diverse types of decisions, it is typical to solve it using decomposition into macro-level planning, which is a combinatorial problem responsible for defining the structure of the plan, and micro-level planning, which elaborates the details of each individual task. In this paper, the decomposition approach presented in (Kardos, Kovács, and Váncza 2017) is adopted, which uses a macro-level planner to optimize the assembly task sequence and the assignment of the resources (tools and fixtures) to the tasks, while a collection of micro-level sub-problem solvers ensures that the planned tasks can be implemented in physical reality. Sub-problem solvers in the current implementation include technological feasibility, collision avoidance, fixturing, and tooling modules.

The solution approach proposed in (Kardos, Kovács, and Váncza 2017) is Benders decomposition with a mixed-integer linear programming (MILP) model applied to solving the macro-level planning master problem and various customized lower-level solvers generating feasibility cuts for the master problem. This approach guarantees the optimality of the macro-level plan and its micro-level feasibility according to all micro-level aspects investigated. The MILP formulation of the master problem is solved using the commercial mathematical programming software FICO Xpress 7.8. The Benders framework, as well as the sub-problem solvers responsible for technology, fixturing, tooling, and collision avoidance, have been implemented in Python. The latter module performs collision detection using the Flexible Collision Library (FCL) (Pan, Chitta, and Manocha 2012) on the STL triangle mesh models of the involved objects.

Assembly Path Planning

Assembly path planning is performed for each task separately at two distinct phases of the workflow: (1) during assembly process planning as a micro-level sub-problem to assess the geometrical feasibility of an assembly sequence, and (2) during motion planning for generating the motion sequence to be actually executed. In both phases, the motion path of the ensemble of moved parts and the attached tool (gripper or an actual tool, e.g., screwdriver), treated as a single solid object, is planned from a remote position to the *near* position of the corresponding assembly feature. The base parts and the fixture are considered to be the obstacles. Planning is performed using an implementation of the classical Rapidly-exploring Random Trees (RRT) algorithm.

In phase (1), when the workcell layout is unknown and the actual resources to carry out the task are not selected

yet, path planning handles the moved parts as free-flying objects with 3 or 6 degrees of freedom (DoF): for most of the assemblies investigated during preliminary experiments, it was found that allowing translational movements results in a feasible path when there exists one. Nevertheless, the path planner can consider 6-DoF problems as well for more complicated geometries. The only answer expected from the path planner in this phase is a yes (a collision-free path has been found) or no (the iteration limit has been hit without finding a collision-free path) answer, and the quality of the path is disregarded.

In contrast, at the phase of motion planning, the goal is to generate paths that will be actually executed. Here, the details of the method differ for robots and for humans. For tasks performed by humans, which is the focus of the present paper, path planning is still performed for free-flying objects (hence, it is implicitly assumed that where parts and tools fit, there the human worker can also access the location of the task). However, path smoothing is applied to the results of RRT to arrive at a path which looks reasonable to workers on animated work instructions. For a robotic task, path planning is performed in the robot's joint space, taking into account the results of workcell layout design, including the pick-up and put-away locations of the parts and equipment (the latter function is not fully implemented yet).

Generating and Presenting Work Instructions

Assembly systems are moving towards using digital technologies for handling and dispatching work instructions. Digital WISs offer, on the one hand, consistency and maintainability for the content, while on the other hand, they also provide a wide set of multi-modal, interactive input/output interfaces. In a modern changing and complex assembly production environment, it is also important that the WIS is connected to the execution control and supervisory systems, which provide information regarding the shop floor status, thus tailoring the delivered information to the actual context.

The approach of the paper follows the workflow shown in Fig. 3, which explains how WIS is involved in the generation and the context aware delivery of instructions. The aim of automated instruction generation is to create animated 3D and textual content. An assembly process engineer can use this content as a basis for finalizing the instructions (by tailoring or extending them with additional content, e.g., videos, pictures, etc.). However, during instruction generation, it has to be taken into consideration that workers may have different skills, language proficiency and content delivery devices at hand. These properties define the part of the worker context which needs to be addressed during instruction generation. After content generation, instructions are delivered by the WIS according to the complete context, which is defined by the following aspects:

- Process context: the task to be executed, which triggers the instruction delivery, coming from execution tracking.
- Worker activity context: actions performed by the workers which form commands toward the execution tracking system. The commands (e.g., acknowledgement, failure) are interpreted by the WIS and sent to the execution tracking system.
- Worker location context: actual position of the worker in the shop floor, which identifies the worker in the workcell, thus enables delivering location-aware instructions (e.g., increasing font size or volume when the worker is further away from the devices).
- Worker properties context: skills, preferences of the worker. This information, stored in the worker database, has to be taken into account during instruction generation and delivery in order to ensure that the displayed instructions match the requirements of the given worker.

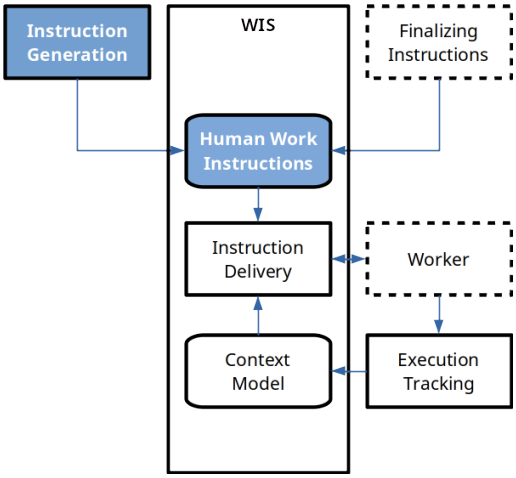


Figure 3: Overview of WIS's role in generation and context-aware delivery of instructions. The connection to the workcell configuration is highlighted in blue.

For the automatic generation of work instructions the features serve as basic templates of instructions. Each individual feature-based task defines the micro-world of its execution. In the presented approach, instructions are skill level-dependent: more skilled workers only require the core instructions (highlighted in bold in the template below). The following hierarchical template structure is used for generating instructions from the feature-based plan representation:

1. simple_feature_task(feature, base_part, moved_part, skill)
   1.1. approach(location(moved_part)) - Approaching location
   1.2. pick_up(moved_part) - Picking up part
   1.3. **core_task(base_part, moved_part, type(feature))** - Performing the core corresponding to the feature type
      * insert(base_part, moved_part) - Inserting moved part into base part
      * place(base_part, moved_part) - Placing moved part to base part
      * screw(base_part, moved_part) - Inserting and tightening screw (moved part) into base part
2. changeover_task(old_equipment, new_equipment), [part], skill)

2.1. release(old_equipment) - Releasing installed equipment (tool or fixture)
2.2. pick_up(old_equipment)
2.3. approach(pick_up_location(old_equipment))
2.4. put_away(old_equipment)
2.5. approach(pick_up_location(new_equipment))
2.6. pick_up(new_equipment)
2.7. **install(new_equipment, [part])** - Install tool / Grasp part in fixture

3. composite_feature_task({simple_feature}, ordering, skill)
   - ordering - Specifying sequence for features inside composite features
     * *parallel*
     * *sequential*
     * *arbitrary*

3.1. **composite_header(simple_feature_task(), ordering)** - Performing multiple tasks in composite feature based on the first task according to ordering
3.2. repeat(simple_feature, ordering) - Performing a list of simple features according to ordering

For the above hierarchical approach to work, the following helper functions are required:

- subassembly(part) - Returns the subassembly to which the part belongs
- pick_up_location(object) - Returns the pick-up location of the object
- put_away_location(object) - Returns the put away location of the object
- type(simple_feateure) - Returns the type of the simple feature

Generating Textual Instructions

Textual instructions traditionally serve as the elementary method for delivering information to workers. The paper-based and the newer digital approaches are both limited by the human effort required to keep them up-to date, which opens up potential for automated or semi-automated textual instruction generation.

Textual instructions are generated for task execution by using the following instruction templates and the data from the tasks:

- approach(location): "Go to *_location*"
- pick_up(object): "Take *object* [in subassembly *subassembly(object)*]"
- insert(base_part, moved_part): "Insert part *moved_part* [in subassembly *subassembly(move_part)*] into *base_part* [in subassembly *subassembly(base_part)*]"
- place(base_part, moved_part): "Place part *moved_part* [in subassembly *subassembly(move_part)*] to *base_part* [in subassembly *subassembly(base_part)*]"
- screw(base_part, moved_part): "Insert screw *moved_part* into *base_part* [in subassembly *subassembly(base_part)*] and tighten it"

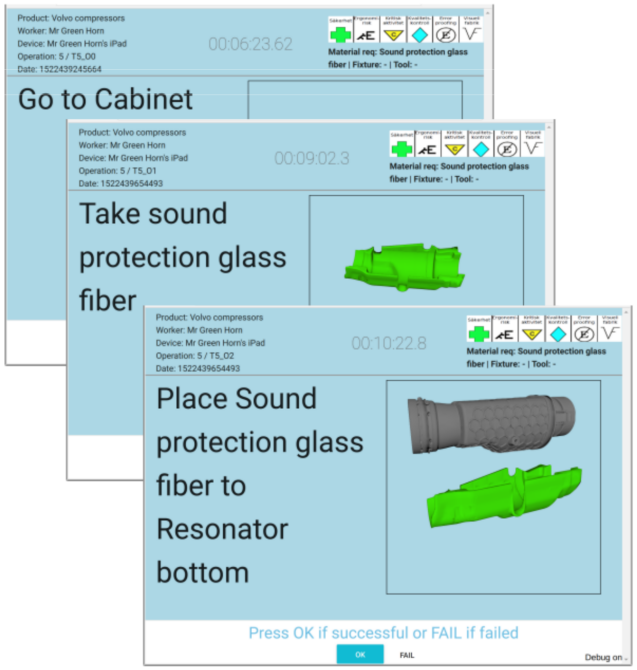


Figure 4: An example of a task delivered to a less skilled worker in more details.

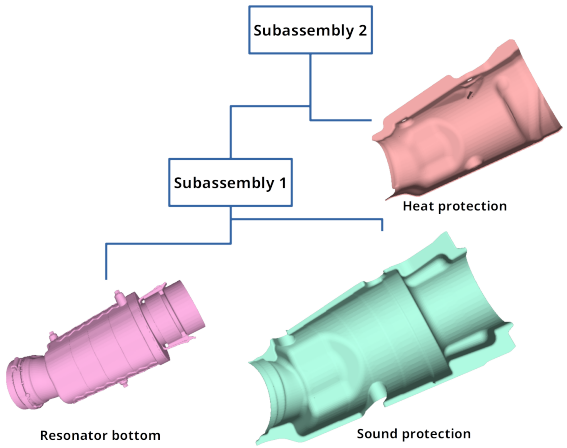
- composite_header(simple_feature_task, ordering): "Execute on all items in the *ordering* order."
- release(object): "Release *object*"
- grasp(fixture, part): "Grasp *part* [in subassembly *subassembly(part)*] in fixture *fixture*"

For workers with advanced skill levels, only the core assembly instructions are delivered using repetition and ordering flags for composite instructions, while for workers in training every step is displayed (e.g., approach, pick-up, core instruction). This is in line with industrial practice, where the supervision of task execution is done on a lower level for workers in training, hence it is required to display and have every step acknowledged. This approach also allows using a single workflow for generating content for both beginner and experienced workers.

Generating Visual Instructions

Micro-level planning enables using the geometrical models in generating animated instructions as well. In order to implement this, X3D format was chosen for displaying 3D instruction content.

X3D is an extension of the VRML format and is part of the HTML5 standard, which means content can be displayed and manipulated through web-browsers. It also supports linking geometries to reference frames and thus enables building up kinematic chains. Using this approach the assembly tree and its components are represented in a multiple level reference frame hierarchy. Additionally, rearranging the nodes of the model is done through the HTML's Document Object Model (DOM) (e.g., attaching the model of




```
<X3D>
 <Scene>
  <Transform DEF="Subassembly 2">
   <Transform DEF="Subassembly 1">
    <Transform DEF="Heat protection">
     <Shape><IndexedTriangleSet ... ></Shape>
    </Transform>
    <Transform DEF="Resonator bottom">
     <Shape><IndexedTriangleSet ... ></Shape>
    </Transform>
   </Transform>
   <Transform DEF="Sound protection">
    <Shape><IndexedTriangleSet ... ></Shape>
   </Transform>
  </Transform>
 </Scene>
</X3D>
```

Table 1: A sample assembly tree with three parts and the structure of the corresponding X3D representation.

the tool to the moved object).

The geometric models applied in planning are triangle mesh models, which offer generic and open access representation. These models can be easily translated into the *(indexed) triangle set* representation of the X3D format. Moreover, as the required resolution for displaying instruction is lower than that of planning, in order to enhance the rendering speed, the 3D models are resampled by applying edge decimation algorithms over the triangle meshes (e.g., quadratic edge collapse (Garland and Heckbert 1997), see Fig. 5).

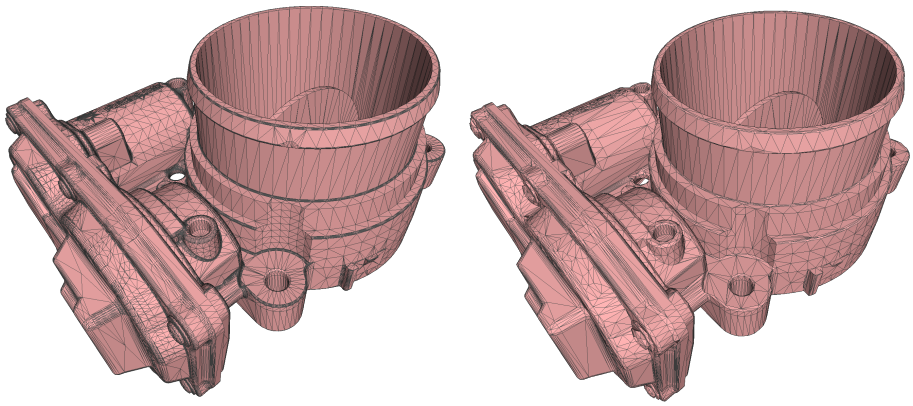
Each geometry in the model (parts, tools, fixtures) are translated into X3D format in such a way that the *shape* node containing the *indexed face set* representation of the geometry is appended to a *transform* node. The *transform* node allows the application of geometric transformation to its children *shapes* by exposing its *translation* and *orientation* fields. Similarly to parts, each node in the assembly tree (i.e., subassembly) is represented by its own *transform* node and therefore contains all the previously assembled parts of its branch (see Table 1).

Building up the X3D scene in a structure symmetrical to the assembly tree enables setting attributes of already assembled parts in either one field at the top of the hierarchy, or by searching recursively via the DOM structure of the X3D-scene (e.g., by using *jquery*). This applies to transformations (position and orientation, exposed by the *transform* node; note that planning assumes that parts once assembled remain in this state) and to display options such as switching rendering and changing color. Hence, the scene (the displayed objects and their position and orientation) can be set to suit a given state defined by an assembly step. Also, the models of tools and fixtures are attached to the corresponding parts or subassemblies dynamically, according to the displayed step. Each assembly step is described by the following data:

- ID of the assembly step
- IDs of the moved parts/subassemblies
- IDs of the base parts/subassemblies
- Transformation of the near position
- Nodes of the path
- ID of the tool
- Tool Center-Point Frame (TCPF) transformation of the tool
- ID of the fixture
- Transformation of the fixture

For displaying 3D instructions two cases are distinguished: (1) showing a static view of components in assembled or disassembled (near position) state; (2) animation of movement provided by path planning. In both cases, it might be required to show only the base and moved components, therefore turning off rendering for all the nodes which are not in the branch of the assembly tree starting from the subassembly node of the actually displayed assembly step. The implemented visual instructions are the following:

- approach(location): static view of location
- pick_up(object): static view of object
- insert(base_part, moved_part): animation of local and approach motion
- place(base_part, moved_part): animation of local and approach motion
- screw(base_part, moved_part): animation of local and approach motion
- composite_header(simple_feature, ordering): visualization according to the underlying simple features and ordering flag
- release(object): static view of object
- grasp(fixture, part): static view of fixture and part



(a) (b)

Figure 5: Two triangle mesh models, representing the same object before (a) and after (b) resampling. By applying a 0.9 reduction the number of vertices were reduced from ˜90k to ˜10k, which has a significant impact on the rendering speed.

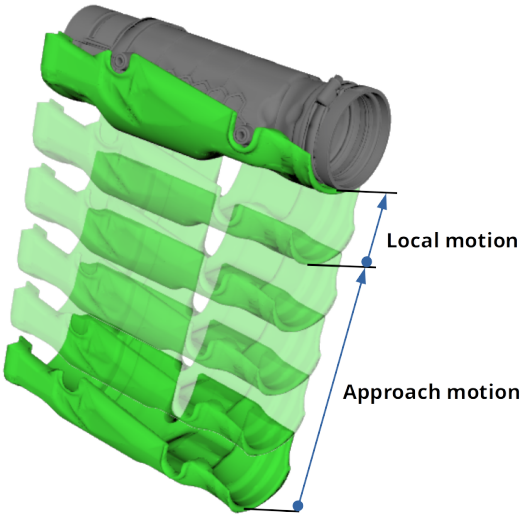


Figure 6: Illustration of X3D animation of two components being assembled, the moved component is shown in green.

As there is no animation in case (1), the assembled and the disassembled state can be displayed by simply applying the transformation of the near position to the *transform* node of the moved component. The transformation applies to all the children nodes (i.e., parts or subassemblies). To create animated movements along the nodes of the path, the *Interpolator* X3D node is utilized, which implements linear interpolation between each node (see Fig. 6). The duration of the interpolation is calculated assuming a constant speed along the path.

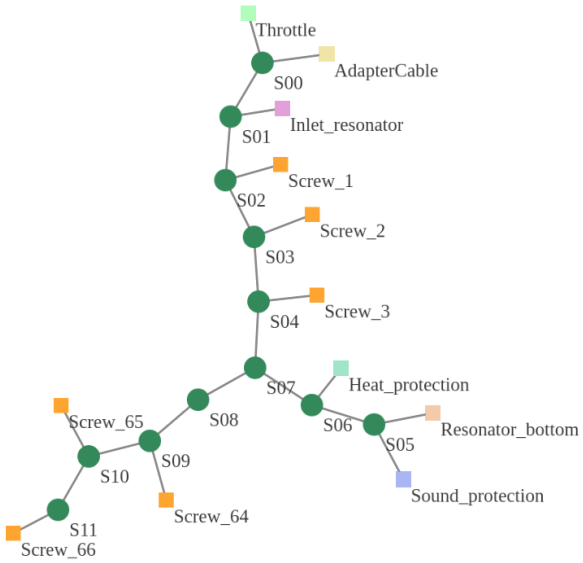


Figure 7: Assembly process plan for the case study, showing the assembly steps from S00 to S11. Parts in composite features (i.e., screws) are highlighted in orange.

Case Study

The proposed approach is demonstrated in a case study from the automotive industry. The assembly, namely the *inlet bypass* is part of a so-called *supercharger* component composed of 29 parts. The *inlet bypass* is built up by 12 parts: 6 main components and 6 screws (see Fig. 8). There are five simple features and two composite features, either of them involving joining a set of parts by 3-3 screws, and one auxiliary task. Solving the CAPP model of the problem resulted in the assembly sequence shown in Fig. 7, which minimizes the time of tool and fixture changeovers.

The demonstration is part of a research project aiming to develop a WIS for supporting multi-modal and context-

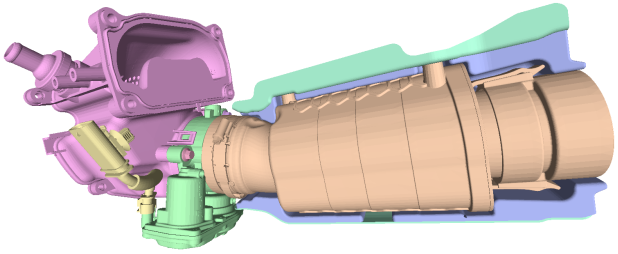


Figure 8: The assembly of the case study contains 12 components (6 screws). The coloring of the parts is similar to that in the assembly process plan (Fig. 7), except for the screws in the composite features.

aware instruction delivery in flexible workcells. The developed system is connected to a workcell-level controller, which controls and tracks task execution and triggers instruction delivery to multiple devices. The WIS has a frontend-backend architecture, where the content database of the backend is populated with the automatically generated instructions based on the CAPP solution. The frontend is responsible for delivering the instructions to the devices of the worker (e.g., smartphone, tablet, screen, headphones). The visual instructions are delivered through a responsive HTML5 web-page, which also utilizes a text-to-speech library for the audio delivery of the textual instructions.

In the case study, instructions were generated for workers with two different skill levels. One is the worker with advanced skills, who receives only the core instructions, the other is the worker under training, who receives more detailed instructions. Figs. 4 and 9 show screenshots of the WIS frontend. In addition to displaying the generated instructions, the WIS frontend can also be configured to display safety or quality symbols (see the upper right corner of Figs. 4 and 9). The research project is now in its closing phase, where the focus is on the development of demonstrator case studies and the evaluation of the perceived work experience with using the generated content and the multimodal content delivery system.

Conclusions

This paper presented a novel approach to the automated generation and the presentation of context-sensitive work instructions for human workers in mechanical assembly. The methods have been implemented and integrated into a module of a comprehensive assembly workcell configuration toolbox, to visualize the assembly plans generated by a mixed-initiative, optimizing process planner. The generated work instructions contain both textual instructions, delivered either visually on a screen or as an audio stream using text-to-speech tools, as well as X3D animations. The main advantage of the approach is the ability to efficiently generate and present customized, context-sensitive instructions that take into account the workers' skill levels, language, devices and presentation preferences.

In order to proceed from the current, prototype-level implementation to an industrial deployment, future work must

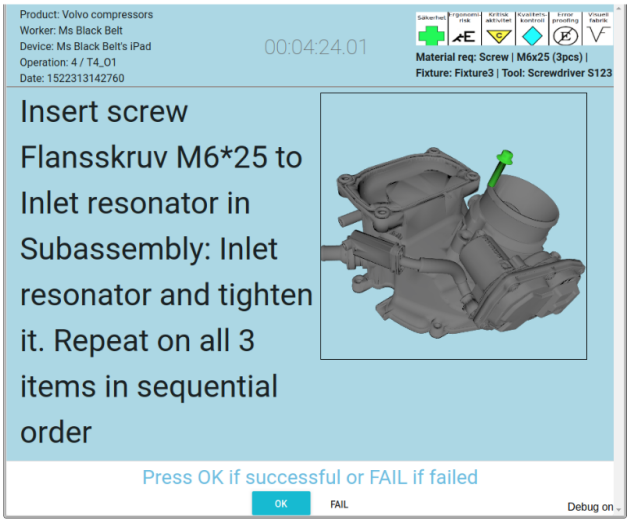


Figure 9: An example of a composite task delivered to an expert worker using the repetition and ordering flags.

focus on extending the instructions to satisfy all relevant industrial standards, such as placing identifiers and safety symbols on the instruction screens, preferably in an automatically generated way. In a wider context, the further development of various planning functions in the overall assembly workcell configuration toolbox holds numerous research challenges. Above all, a stronger support and integration is required for the generation of the feature-based assembly model from legacy design representations.

Acknowledgements

This research has been supported by the GINOP-2.3.2-15-2016-00002 grant on an "Industry 4.0 research and innovation center of excellence" and by the EU H2020 Grant SYMBIO-TIC No. 637107.

References

Bader, S., and Aehnelt, M. 2014. Tracking assembly processes and providing assistance in smart factories. 161–168. SCITEPRESS—Science and and Technology Publications.

Bannat, A.; Wallhoff, F.; Rigoll, G.; Friesdorf, F.; Bubb, H.; Stork, S.; Müller, H. J.; Schubö, A.; Wiesbeck, M.; and Zäh, M. F. 2008. Towards optimal worker assistance: A framework for adaptive selection and presentation of assembly instructions. In *Proceedings of the 1st international workshop on cognition for technical systems, Cotesys.*

Claeys, A.; Hoedt, S.; Landeghem, H. V.; and Cottyn, J. 2016. Generic model for managing context-aware assembly instructions. *IFAC-PapersOnLine* 49(12):1181–1186.

Fiorentino, M.; Uva, A. E.; Gattullo, M.; Debernardis, S.; and Monno, G. 2014. Augmented reality on large screen for interactive maintenance instructions. *Computers in Industry* 65(2):270–278.

Garland, M., and Heckbert, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of the*

24th annual conference on Computer graphics and interactive techniques, 209–216. ACM Press.

Ghandi, S., and Masehian, E. 2015. Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design* 67-68:58–86.

Hu, S.; Ko, J.; Weyand, L.; ElMaraghy, H.; Lien, T.; Koren, Y.; Bley, H.; Chryssolouris, G.; Nasr, N.; and Shpitalni, M. 2011. Assembly system design and operations for product variety. *CIRP Annals* 60(2):715–733.

Kaipa, K.; Morato, C.; Zhao, B.; and Gupta, S. K. 2012. Instruction Generation for Assembly Operations Performed by Humans. In *32nd Computers and Information in Engineering Conference, Parts A and B*, volume 2, 1121–1130. ASME.

Kardos, C.; Kovács, A.; and Váncza, J. 2016. Towards feature-based human-robot assembly process planning. *Procedia CIRP* 57:516–521.

Kardos, C.; Kovács, A.; and Váncza, J. 2017. Decomposition approach to optimal feature-based assembly planning. *CIRP Annals—Manufacturing Technology* 66(1):417–420.

Leu, M. C.; ElMaraghy, H. A.; Nee, A. Y.; Ong, S. K.; Lanzetta, M.; Putz, M.; Zhu, W.; and Bernard, A. 2013. CAD model based virtual assembly simulation, planning and training. *CIRP Annals—Manufacturing Technology* 62(2):799–822.

Lušić, M.; Schmutzer Braz, K.; Wittmann, S.; Fischer, C.; Hornfeck, R.; and Franke, J. 2014. Worker information systems including dynamic visualisation: A perspective for minimising the conflict of objectives between a resource-efficient use of inspection equipment and the cognitive load of the worker. *Advanced Materials Research* 1018:23–30.

Lušić, M.; Fischer, C.; Bönig, J.; Hornfeck, R.; and Franke, J. 2016. Worker information systems: State of the art and guideline for selection under consideration of company specific boundary conditions. *Procedia CIRP* 41:1113–1118.

Michalos, G.; Makris, S.; Spiliotopoulos, J.; Misios, I.; Tsarouchi, P.; and Chryssolouris, G. 2014. ROBO-PARTNER: Seamless human–robot cooperation for intelligent, flexible and safe operations in the assembly factories of the future. *Procedia CIRP* 23:71–76.

Pan, J.; Chitta, S.; and Manocha, D. 2012. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation*, 3859–3866.

Wang, L.; Keshavarzmanesh, S.; and Feng, H.-Y. 2011. A function block based approach for increasing adaptability of assembly planning and control. *Int J of Production Research* 49(16):4903–4924.