Flexible Robot Programming using Solid Edge's "Alternative Assemblies"

Article · February 2022

DOI: 10.30958/ajte.9-1-1

2 authors, including:

Norman Urs Baier
Bern University of Applied Sciences
6 PUBLICATIONS **11** CITATIONS

SEE PROFILE

Athens Journal of Technology and Engineering - Volume 9, Issue 1, March 2022 – Pages 9-24

Flexible Robot Programming using Solid Edge's "Alternative Assemblies"

By Norman Urs Baier[*] & Joel Costan Zovi[±]

Many assembly processes in small and medium-sized enterprises are still performed by human labour. One reason for this is the need for another expert to program the robot, which would simply not fit into the company structure. To address this issue a solution is developed, which allows to program the robot directly out of the CAD software. The positions of the parts are read out of the CAD file. Specific assembly instructions have to be given by the assembly developer and integrated in the tree structure of the CAD. To avoid collisions and ensure correct insertion angles, additional waypoints are given by alternate assemblies, a functionality within Solid Edge to create and use variations of an assembly.

Keywords: *assembly, task planning, intelligent and flexible manufacturing, CAD*

Introduction

The classical use of an industrial robot is within a repetitive process. It is taught a few positions with the teach panel or by other means and then it starts doing its task. If the process is a manufacturing or assembly process, then most often a single robot performs tiny subtasks of the whole process and most often the production volume is high. Markis et al. (2016) distinguish four different production paradigms, namely fixed automation, robotic automation, human-robot collaboration, and manual assembly. As production costs do not scale with lot size for manual assembly this is the solution for smallest lot sizes, whereas fixed automation is for rather large lot sizes. For lot sizes in between, though, robots may also be deployed profiatinerly. Depending on lot size and complexity of the manufacturing process different strategies have been developed.

For some manufacturing processes of more complex nature, the robot will need to cowork with a human operator even in the foreseeable future. This situation requires different programming paradigms than situations in which the robot alone is capable of completing the tasks, but some single elements change after very small batch sizes. Wang et al. (2019a) have identified and named four different forms of human-robot relationships: coexistence, cooperation, interaction, and collaboration. Clearly, in case of collaboration the robot needs to understand the human and hence appropriate programming paradigms need to be available. These need to include on one hand the ability to learn quickly from human interaction

[*]Professor for Control Engineering and Mechatronics, Bern University of Applied Sciences, Switzerland.
[±]Microcut Ltd, Switzerland.

https://doi.org/10.30958/ajte.9-1-1 doi=10.30958/ajte.9-1-1

despite possible ambiguity (Thomaz and Breazeal 2008, Wang et al. 2019b) and on the other hand they need to establish possible ways the human can express himself toward the robot (Tsarouchi et al. 2016, Cserteg et al. 2018).

In contrast, many manufacturing processes performed in small and medium sized enterprises (SME) could be performed by a robot without the help of a human co-worker. There are several reasons why often manual labour is preferred in SMEs. One of them is that classical robots take up too much space, because they need to be behind fences (Perzylo et al. 2016), a problem solved to a large extent with collaborative robots in a coexistence scenario. Another reason is the programming effort and the apprehension an expert might be needed to perform the programming (Perzylo et al. 2016). The usability of the user interfaces of collaborative robots has been analysed by Schmidbauer et al. (2020). To address the need for effortless deploying of robots, software has been proposed by fortiss, for example: Perzylo et al. (2019) describe a software which allows to program a robot by manipulating physical parts and drawings of them on screen. The assembly itself can be divided in subtasks, each of which is capable of performing a particular action. These are called skill primitives and depending on robot and sensors installed, different skill primitives can be realised (Watson et al. 2020).

Alternatively, manufacturing or assembly data can be extracted directly out of existent CAD files; an approach which has been identified as promising by von Drigalski et al. (2020). Known in literature is a strategy called "assembly by disassembly", it proved to work for the assembly of different types of housings (Michniewicz et al. 2016). Recently, strategies have been published in which the tree structure in the CAD file is used to harbour instructions for assembly (Linnerud et al. 2019, Transeth et al. 2020).

A link still not duly carved out to or opinion is how existing work processes in SMEs may be altered such that in the end a robot can assemble the designed item. Particularly, it may not be necessary or not even desired that the assembly sequence is generated automatically. Instead, usually the product designer has a clear idea already how the item shall be assembled, and the robot should do, the way the designer intended it to go on. Hence, it should blend in with existing development and design processes, which most often are amended CAD files. Furthermore, to increase the acceptance of robots in SMEs the robot itself should blend in with existing workplaces traditionally designed for humans. It should be able to use those tools, which humans use as well. In this paper we present a set-up to perform assembly with a robot, which addresses the problems within the above-mentioned frame.

Bin picking, on the other hand, is not considered as within the frame: all objects and items are provided at known locations. Provisioning is an important part in a complete assembly process; however, it can be solved detached from the actual assembly process.

Methodology

An example item is assembled with an off-the-shelf collaborative robot. In

this section we give all the details of the example item, the robot, the tools, and software, which were used for the task.

Example Item: Spur Gear Unit

To develop and assess the automatic programming, an example item was selected. The requirements for the example item were:

- size such that an ordinary collaborative robot can handle it,
- composed of a manageable number of parts,
- different assembly tasks involved during assembly,
- preferably an item, which is publicly available for purchase.

Figure 1. *Exploded Drawing of the Spur Gear Unit*

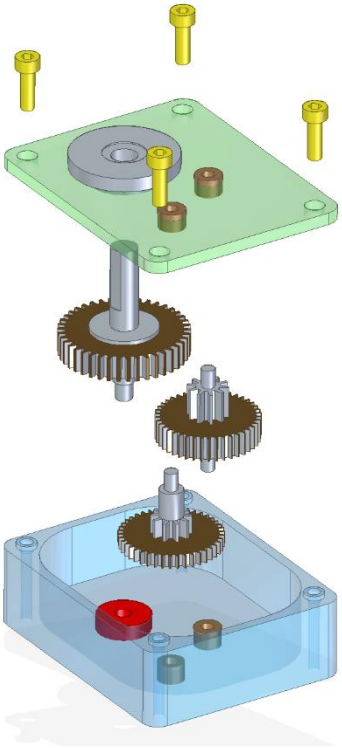
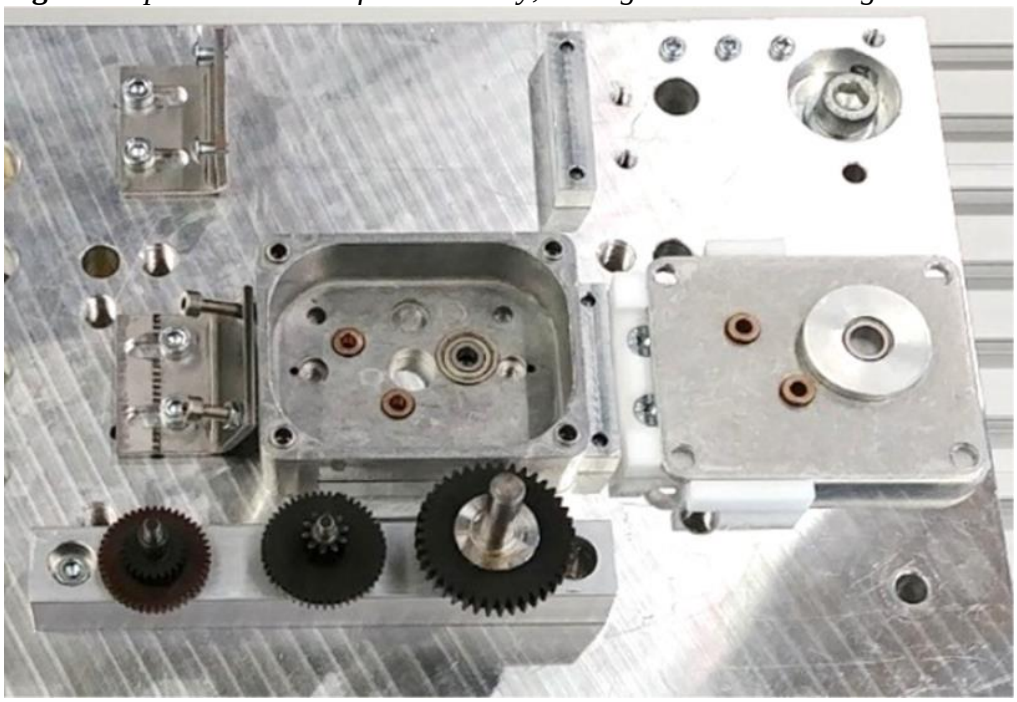


Figure 2. *Spur Gear Unit before Assembly, Arranged on the Mounting Plate*



The choice fell on a spur gear unit of Hilba, more precisely the model GOBUN5713FR100-01. It consists of a housing with bearings, 3 gearwheels on axes fitted to the bearings and a lid screwed on the housing. Its assembly consists of inserting the gearwheels into the bearings. For the second and third gearwheel the correct interleaving has to be observed. Furthermore, posing of the lid is an involved process. It includes three steps and screwing. The lid comprises the bearings for the axes of the gearwheels. First, it has to be posed and then the correct insertion of all three axes has to be verified. For the screwing of the lid, a tool (screwdriver) needs to be used. Figures 1 and 2 show the spur gear unit in a disassembled state and as an exploded drawing.

Robotic Arm, Gripper, Sensors and Tools

There are no particular requirements for the robot, on the contrary, the less particular the robot is, the better the results can be ported to other installations. For reasons of availability, the choice fell on a F&P Robotics P-Rob 2. This is a 6 DOF industrial robot with a straight idle position. It is originally equipped with a servo gripper, but for this work a custom hand with three separate pneumatic grippers is installed.

Furthermore, the HEX-E force torque sensor of On Robot A/S is installed on the robotic arm and an electric screwdriver is mounted within the reach of the robotic arm. The screwdriver can be switched on and off through the digital connections of the robotic arm and hence controlled through the proprietary robot software myP (F&P Robotics 2018a, Mišeikis et al. 2020).

Communication with and triggering of routines within the robotic arm was

done through a TCP-socket. Through the socket myP provides possibilities to trigger functions and procedures written in Python. With the help of those the adoption of poses can be triggered, and the digital IO-connections can be read or set. myP offers an extensive set of functions to do so (F&P Robotics 2018b). The functions that are used in the proposed realisation are:

- `write_digital_outputs()`: to control attached electrical equipment like the screwdriver or the pneumatic valves to close the grippers,
- `read_digital_inputs()`: to observe the feedback on torque and general errors of the screwdriver,
- `run_advanced_path()`: to make the robotic arm move like required by the assembly program,
- `read_tcp_pose()`: to read out the current position of the tool center point and correspondingly continue with the assembly.

Other functions like `close_gripper()` and `open_gripper()` are also implemented in myP, but have not been used. These two functions in particular because other grippers have been installed. No myP-functions are used to read out the HEX-E force torque sensor, it is connected directly to the PC running the assembly program.

CAD Software

In order to blend in as good as possible with existing workflows, the interface to the CAD-data is set up on application layer, not on file layer. This means Step-files or similar files are not considered as input source, but an appropriate and existing API towards a widespread CAD software suite is used. Here Solid Edge was used for the implementation; among the more important softwares according to market share (Warfield 2020), it proved to be most easy to interface to: Through the libraries *Interop.SolidEdge* and *SolidEdge.Community* data contained in a Solid Edge design can be extracted and manipulated in a straightforward manner from any C# software project.

Product Development and Assembly Development

For our method we started from the working assumption that product development and assembly development are two separate steps in the workflow of the manufacturer. The method does not aim at eliminating assembly development by the help of artificial intelligence or other means. So, for the assembly itself it is assumed that the assembly developer performing the actual development will receive all data concerning the product to be assembled in form of a CAD file and that the technician then starts to plan in which order the individual components need to be aligned and merged and what particular steps are involved.

In fact, assembly development is always involved in conjunction with product development, however it can take a different appearance depending on the company carrying out the product development. The method presented here

requires the assembly development to be carried out in the CAD software as opposed to more manual workflows involving text documents and photos of an example assembly, which is still very common in smaller companies.

Performing the assembly development in CAD is advantageous for a work process where assembly development is performed by a constructing engineer and most advantageous if it is performed right after product development by the same engineer, who has performed the product development. The engineer can then exploit all the skills already used during design phase. Such a process would ideally fit very small companies.

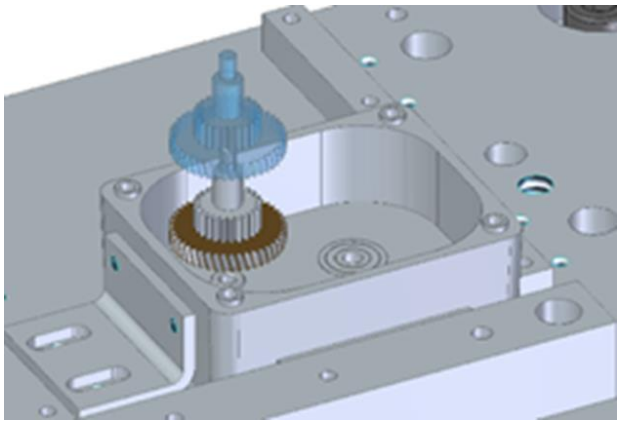
Results

In the current implementation a software procedure written in C# (the program) connects to the CAD data on one side and the robotic arm on the other side. Before focusing on how information is processed and transmitted from one element of the installation to the other, it is important to specify some of the details of Solid Edge, which are important for the implementation.

Alternate Assemblies for Additional Waypoints or Movements

In common industrial workflows the assembly developer has the crucial task to assure that gripping the part to be added to the assembly is possible and the joining position can be reached with the defined grip, independent of the fact if the assembly is performed by humans or robots. To verify if the grip is possible, the gripper is represented in the CAD tool. To define the actual joining movement, additional waypoints may be necessary. Here, alternate assemblies are used to define these waypoints. "Alternate assemblies" is the name of a functionality within Solid Edge, which can be used to manage variations of an assembly (Siemens Product Lifecycle Management Software Inc. 2011). It allows to store an assembly in different configurations, showing the parts, out of which it is assembled, in different locations and orientations. It can also be used to store configurations, in which some parts differ. Without this functionality, it would be less convenient to mark parts as identical and keep the different variants of the assembly in place. With this functionality, the presence and location of the assembled parts can be stored in a single file for different configurations. Within Solid Edge, the different assembly configurations are called members.

Figure 3. *Member2 Showing Two Different Intermediate Positions of the First Gearwheel*

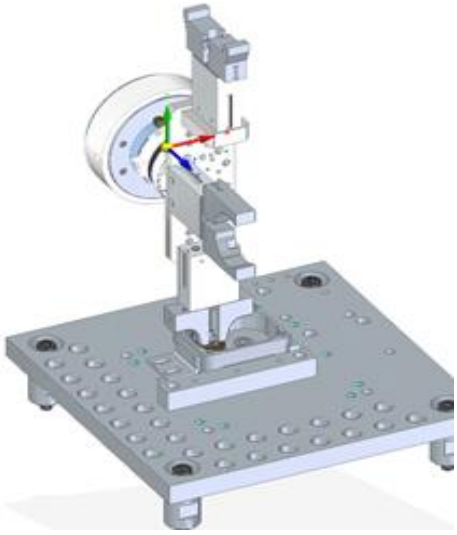


In our implementation the members are used to define subsequent poses of the element to be joined. Hence, in one member the assembly is shown with the part currently to be joined in its final location, while it is shown in other members in intermediate positions. Figure 3 shows two different alternate assemblies in an overlay: In one alternate assembly (shown in transparent blueish) the first gearwheel has an intermediate position, in another (shown solid) it has its final position. With these means, the assembly developer can harness the robot very precisely. Furthermore, through commands written into the placement name (described below) the program can be informed that a particular skill is necessary to complete the assembly step. Instancing the step shown in Figure 3, most often mounting an axis into its bearing cannot be done with a simple linear movement but requires a sequence of movements and measurements. One way to give the robot the ability to exhibit more elaborate behavior than simple linear movements is skills (Thomas et al. 2003). The skills, which are necessary to assemble the spur gear unit, and hence are now already implemented in the program are also described below.

Individual parts in the CAD file can have a different origin. It is common practice to use third party parts in assemblies and import the corresponding part from a CAD library of the supplier of the part and the library. Therefore, the origin of the coordinate system of a part cannot reliably be used to determine the gripping pose, because the supplier can put it anywhere. As a solution the gripper is introduced into the CAD drawing. The task of the assembly developer becomes then to align the gripper with the part to be handled, which can be done with few clicks by a trained designer. Display the gripper in the assembly is recommended anyway, as it also allows to check for collision free gripping and releasing while planning the assembly procedure. The assembly showing the part to be handled together with the gripper is shown in Figure 4.

In our implementation the numbering scheme of the members and the parts within the members are used to specify the assembly order.

Figure 4. *Member1 Showing Gripper Mounting Plate and Final Joining Position of the First Gearwheel*



Interfaces & Structure

Figure 5 shows all involved elements of the installation used in the current implementation. The central element is the above-mentioned C# program, which connects

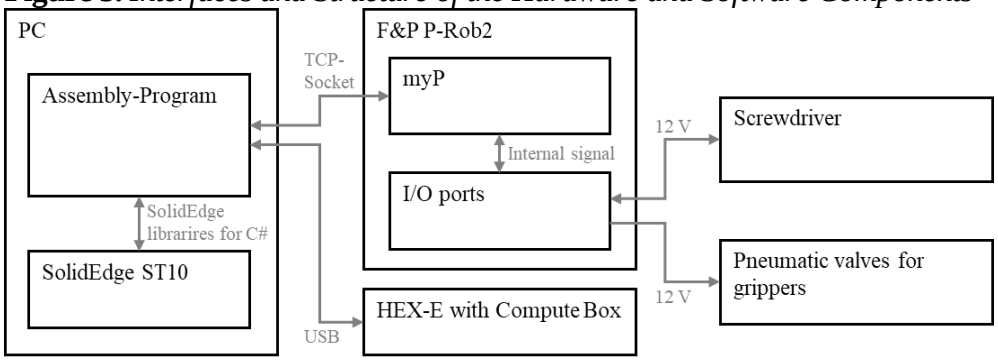
- to Solid Edge with the help of the corresponding libraries,
- to the software "myP" on the robotic arm through TCP sockets, and
- to the force torque sensor through its USB interface.

The robotic arm connects

- to the electrical screwdriver with its control box, and
- to the valve controlling the three pneumatic grippers,

both through its digital IO ports.

Figure 5. *Interfaces and Structure of the Hardware and Software Components*



The interface of the program to the robotic arm and accessories is straightforward: Through this channel merely pose and gripping instructions are transmitted in an implicit way by triggering the functions described in the previous section. The data processing, which breaks down the complex assembly task to single poses to reach and gripping instructions, takes place beforehand when the program reads out the instructions of the assembly developer stored in the CAD file.

Instructions in the CAD File

During assembly development, necessary information to successfully perform the assembly is created. This information is stored in the CAD file. This includes information on possible tools that are used during the joining, where individual parts are located and the pose with which the gripper has to reach for those parts.

Solid Edge provides a "placement name" to describe a particular entity of the component used. It can be user defined and here, it is used to store information on the assembly process. The existing placement name is augmented with key-value pairs describing the action that needs to be performed. All implemented keys are shown in Table 1. The key-value pairs are separated by a semicolon. How this looks in Solid Edge is shown by the screenshot in Figure 6.

Figure 6. *Screenshot Showing the Highlighted Member in the Edgebar and the Pathfinder with Parts Having a Custom Placement Name*

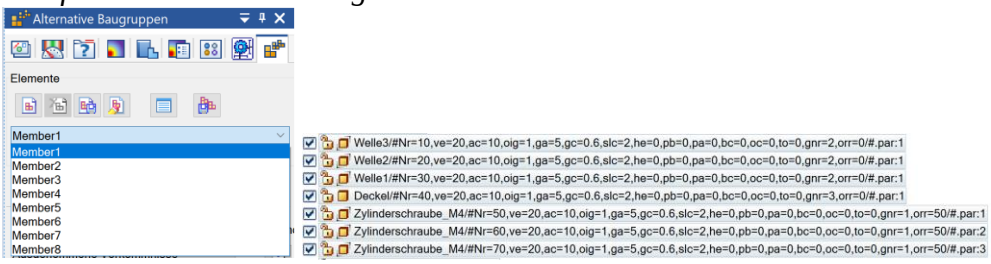


Table 1. *Possible Qualifiers in Placement Name*

Key	Meaning
`nr`	Sequential number of the assembly step
`ve`	Velocity
`ac`	Acceleration
`oig`	1: Gripping outward, 2: Gripping inward
`he`	Tool use, 1: Screwdriver
`pb`	Intermediate position
`oc`	Gripper open or closed, 1: Open, 2: Closed
`to`	Skill primitive, 1: bolt into bearing, 4: gearwheel
`gnr`	Gripper number
`orr`	Retraction

The order in which parts have to be gripped and joined is specified by the value given with the key `nr`. Furthermore, `ve` and `ac` keys can be given to limit velocities and accelerations during that step. Whether or not the part has to be

gripped outward or inward is specified by key `oig`.

In case the part is not gripped or handled directly by the gripper, the key `he` is used to specify which tool has to be used. Currently only the screwdriver is implemented. With the key `pb` intermediate positions can be specified: The part will be moved successively from the position with the highest `pb` number to the position with the lowest `pb` number. Hence, a trajectory with angles can be forced. This may be necessary to fit the gearwheels or the lid onto the assembly, or whenever elements of the assembly have to be avoided during joining. The key `oc` allows to close or open the gripper at particular position.

To specify how the joining has to be performed, the key `to` is used. With this key, the program is told what substeps are involved during the joining. Currently two different situations are implemented: A bolt has to be fit into a bearing (value 1) and a gearwheel has to be interlocked (value 4). The solutions involve skill primitives and are detailed in the next section.

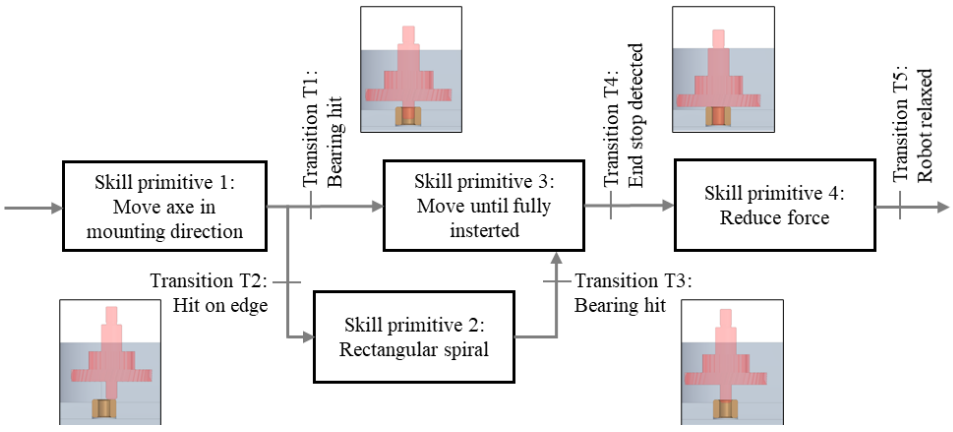
Which gripper shall be used to grip the part can be specified with the help of the key `gnr`. The key `orr` is most useful in conjunction with the key `pb`. With its help a retraction distance can be specified. The gripper is then retraced for the specified distance along the final axis and does not rewind the path specified by the intermediate positions given by `pb`.

Skill Primitives

The skill primitives and the associated transitions are usually organised in nets like the one shown in

Figure **7** (Thomas et al. 2003). In the context of the assembly considered here, skill primitive nets were created for mounting axes, interlocking gear wheels and mounting the cover lid. In these tasks the skill primitive nets are a way to handle the uncertainty, which could prevent the task from finishing successfully otherwise. They are triggered through particular commands described above. When the assembly developer declares for example that the current part is an axle, which should go into a bearing, then the program calls the associated skill primitive net implemented in the C# code.

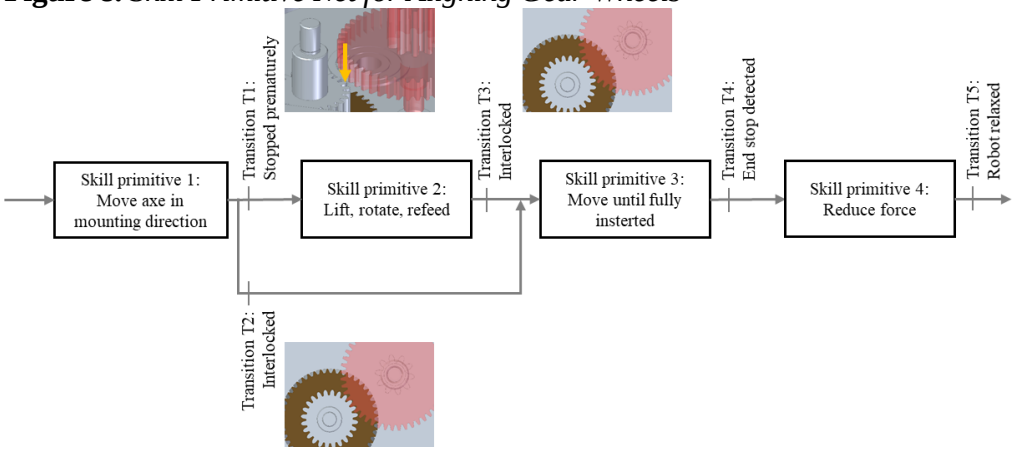
Figure 7. *Skill Primitive Net for Mounting an Axle*



The skill primitive net for mounting an axle is composed of four skill primitives. A block diagram for it is shown in Figure 7.

Skill primitive 1 involves moving the axle along the mounting direction. Either until a critical depth is reached, or the force measurement increases. More precisely, if the measured position exceeds the position, which can be reached when the axle hits the edge, the transition T1 is taken and skill primitive 3 "move the axle vertically until fully inserted" is activated. Otherwise, transition T2 is taken and skill primitive 2 "moving the axle in the shape of a rectangular spiral" is activated first, before skill primitive 3 is activated via transition T3. This happens when the vertical force decreases significantly. The last skill primitive in this net is "reduce inserting force" and the transition between skill primitives 3 and 4 occurs when force measurements increase while position measurements stall. The skill primitive net is left when the force measurement confirms the relaxed pose.

Figure 8. *Skill Primitive Net for Aligning Gear Wheels*



When gear wheels are successively assembled in a spur gear unit, the subsequently added gear wheels have to be aligned to the previously mounted gear wheel. For this situation we created another skill primitive net, which is shown in Figure 8. The outline is quite similar to the first skill primitive net for mounting an axle. In our application the skill primitive net for mounting an axle is always executed before the skill primitive net described here. The defined skill primitives are:

1. move gear wheel in axle direction,
2. lift, rotate and press gear wheel in axle direction again,
3. move in axle direction until final position is reached,
4. reduce inserting force.

The possible transitions are from skill primitive 1 to skill primitive 2 when force measurements increase while not having reached the required vertical position or to skill primitive 3 in case the gear wheels were already aligned and the vertical position was reached without a significant increase in vertical force the same criterion forms the transition from skill primitive 2 to skill primitive 3 and

the transition from skill primitive 3 to 4 occurs when the force increases while the position measurement is reached the final state. The skill primitive net is left from skill primitive 4, when the force measurement has reached a relaxed state.

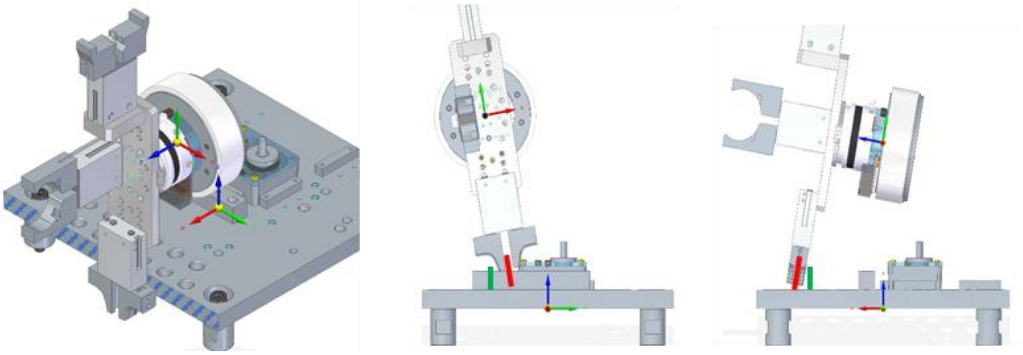
Calibration of the Mounting Plate

In our example, the assembly takes place on a mounting plate. It serves as a reference for all parts that take place in the assembly. The different joining operations during the assembly require the positions of the joined part to be known with an appropriate precision. The worse the precision is, the longer the joining can take, as the robotic arm has to search for the correct mechanical stops.

The assembly performance benefits when the physical pose of the mounting plate coincides precisely with the pose of its digital representation, such that tolerances in mounting plate and robot links are compensated for either by mechanical manipulation or by calculations. For the purpose of calibration, the mounting plate has a number of holes in it that can be used in the process of calibration. The precise process of calibration we adopted for consists of both a mechanical adjustment and a computational correction.

First the orientation of the mounting plate around its vertical axis is set mechanically. The robot gripper is lowered so that the mounting plate can be oriented flush with the yaw angle of the gripper. Next, rods are used to align the gripper with the mounting plate for pitch and roll angles of the gripper (Figure 9). In the end correction angles for pitch and roll for each gripper are known. These depend heavily on the individual robot and the force-torque-sensor.

Figure 9. *Procedure to Calibrate Mounting Plate: First the Yaw Angle around the Blue Arrow is Adjusted Mechanically (Left, Hatched Surface), then the Pitch Angle Around the Red Arrow (Middle) and the Roll Angle around the Green Arrow (Right) Are Measured with the Help of a Bolt*



With the previous steps the orientation of the mounting plate with respect to the CAD data has been assessed, however, discrepancies in the position data are still possible. To close that gap the robot is again made holding a rod, which this time is inserted into three of the holes in the mounting plate. When inserted in the hole, the robot pose is tuned such that the force measures zero and the corresponding

positions from the robot control are read out and hence the position vectors of the three centres of the holes are known. By simple vector subtraction and normalisation, the coordinate system of the mounting plate can be calculated. Now, given a pose of a part in the CAD system, the corresponding pose of the part on the mounting plate easily calculated.

Sequences of the Program in Chronological Order

To read out the position information and to prepare them for use in the assembly command for the robot, the alternate assemblies are run through in order. The libraries Interop.SolidEdge and SolidEdge.Community are used to access Solid Edge through the API and load the alternate assemblies to read out the position of the part whose turn it is.

The alternate assemblies are named Member and numbered through in our implementation: "`Member1`, `Member2`, …". In the first member (`Member1`) the assembly is drawn in its final assembled state. The assembly program begins by reading out `Member1`. One fundamental position information which is currently stored in `Member1` is the position of the gripper itself: The position of the gripper, when the robot is in its rest position is drawn in `Member1`.

The order of the assembly is given in the placement name by the attribute "`nr`" visible in Figure 6. In steps of 10 this attribute directly gives the order of the assembly, whereas the final assembled position is given in the alternate assembly with the name "`Member1`". If there is another alternate assembly showing the same part in another pose, then the robot has first to reach this pose before it brings the part to the final pose.

In the end of that step all data has been extracted from Solid Edge and has been stored in an array. The data in the array are insertion and final positions and the command string, the assembly engineer had put in the placement name.

To make the data accessible and store them for future use, they are stored in a file in JSON-format.

Next, the position data together with the instructions in the placement name need to be processed, with the aim to generate a sequence of commands the robot understands. The sequence will then be sent to the robot to perform the assembly.

The table previously stored in JSON-format is already ordered according to the succession of the assembly. To get the assembly commands for the robot the command string from the placement name (Table 1) has to be interpreted and broken down to commands for the robot (section Robotic Arm, Gripper, Sensors and Tools). As programming paradigm, we used skill primitives, with which we divided the assembly tasks in subtasks.

Discussion

A complete workflow from product design through assembly development to automated assembly has been implemented and tested. The aims pursued were in order of importance

1. blend in with existing product development workflows,
2. blend in with existing workplaces designed for humans,
3. be flexible and allow for assembly of a large variety of products.

To blend in with existing workflows it was sought to link directly the most common tool for product design and development (the CAD software) and the most common tool for automated assembly (the industrial robot with its control software). Both have been linked through an additional piece of software, a program written in C#. This requires the assembly development to take place within the CAD environment. As such this solution targets companies, who do not perceive this requirement as limiting, and hence it is certainly suited for smaller companies, which do have rather simple processes for assembly development.

The solution employs the placement name of Solid Edge and its member structure, to store the information, which was generated during assembly development. As such it depends on Solid Edge, though it can easily be ported to other CAD tools if the necessary interfaces are available. The solution shows a simple way to augment the raw CAD file with information on how to assemble the unit. It also shows a possible way how tools can be used, which is one requirement if the robot should be able to blend in with workplaces for humans. In the current solution the use of tools is controlled by the corresponding key-value-pair in the placement name. How the tool is used is hard coded in the program. Only the use of a screwdriver has been implemented.

Valuation from industrial partners has been positive. Deleting parts from the CAD assembly and rerun the program will result in the robot not taking up that part during the next assembly procedure. Moving the part to another location will result in the robot picking up that part at another location or performing the joining steps at another location, which will work when the other joining parts are prepared for the new location or bring the system to an error state if for example the fitting is at the wrong location. There are no limits other than those of the robot and the fact that only a limited number of joining operations are already implemented. Likewise, the speed of the assembly procedure is only limited by the usual safety aspects and the speed of the robot. The program execution to extract the data from the CAD and to generate the assembly steps happens virtually instantly.

An important aspect is the skill primitive nets. The level of abstraction of the instruction which can be given as instruction in the tree structure, depends on the implemented skills. With the current implemented skills, the assembly of the gearbox can be accomplished, and expected tolerances can be overcome, however, the instructions for the assembly have to be laid out on a very low level. A more extensive skill set would allow for a more comfortable assembly development experience.

Concerning the flexibility in general, the current solution shows where the biggest open issues are to find. These are most of all gripping issues. In the current solution three pneumatic grippers with fingers with positive-locking grip each dedicated to a particular part are installed. This ensures safe grip and precise

positioning for all projected parts, but fails to handle parts, which deviate from what has been projected by the time of finger design. As alternatives to positive-locking grip regripping (Tajima et al. 2020), visual servoing (Watson et al. 2020), haptic feedback (Chin et al. 2019) and novel gripper configurations (Angelini et al 2020) may be considered. How these can be combined to offer efficient new skills needs to be investigated.

Conclusions

With the proposed software, a robot can perform an assembly without the need for being programmed. In the case of the gripper and parts considered here, a collaborative robot could perform the assembly without fences, however a non-collaborative robot behind fences would be able to perform the assembly faster. The assembly development is carried out by the product designer and not by the robot or its controller. As such the operator still has full control over the assembly process without the need to be trained in robot programming. For a comprehensive control of the robot's movements, alternate assemblies have been used to specify intermediate positions. These proved to be a viable way to avoid collisions and reach the right positions before inserting axes, screws, and other parts, which have to be inserted from a defined angle.

Furthermore, during assembly development no further software or tools are used than the CAD software and the program presented here. This is a first step in making the robots blend in better in existing work processes. With a more versatile set of skill primitives, other types of input sources might be considered. Especially documents in human readable format could boost acceptance of robots SMEs.

The flexibility of the solution is limited by the positive locking grip, which was adopted to reach the precision necessary for the peg-in-hole and gearwheel-aligning tasks. Hence, to improve the method, the research focus on that subject should be intensified.

References

Angelini F, Petrocelli C, Catalano MG, Garabini M, Grioli G, Bicchi A (2020) SoftHandler: an integrated soft robotic system for handling heterogeneous objects. *IEEE Robotics & Automation Magazine* 27(3): 55–72.

Chin L, Yuen MC, Lipton J, Trueba LH, Kramer-Bottiglio R, Rus D (2019) A simple electric soft robotic gripper with high-deformation haptic feedback. In *2019 International Conference on Robotics and Automation (ICRA)*, 2765–2771.

Cserteg T, Erdős G, Horváth G (2018) Assisted assembly process by gesture controlled robots. In *Procedia CIRP*, 51–56.

F&P Robotics (2018a) *Git repository for myP additions to ROS*. Retrieved from: https://github.com/fp-robotics/myp_ros. [Accessed 18 February 2021]

F&P Robotics (2018b) *myP Script Functions Manual Version 1.3.2*. Glattbrugg, Switzerland.

Linnerud ÅS, Sandøy R, Wetterwald LE (2019) CAD-based system for programming of robotic assembly processes with human-in-the-loop. In *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, 2303–2308.

Markis A, Montenegro H, Neuhold M, Oberweger A, Schlosser C, Schwald C, et al. (2016) *Sicherheit in der Mensch-Roboter- Kollaboration*. (Safety in human-robot collaboration). Wien.

Michniewicz J, Reinhart G, Boschert S (2016) CAD-based automated assembly planning for variable products in modular production systems. In *Procedia CIRP*, 44, 44–49.

Mišeikis J, Caroni P, Duchamp P, Gasser A, Marko R, Mišeikiene N, et al. (2020) Lio-A personal robot assistant for human-robot interaction and care applications. *IEEE Robotics and Automation Letters* 5(4): 5339–5346.

Perzylo A, Nikhil S, Profanter S, Kessler I, Rickert M, Knoll A (2016) Intuitive instruction of industrial robots: semantic process descriptions for small lot production. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2293–2300.

Perzylo A, Rickert M, Kahl B, Somani N, Lehmann C, Kuss A, et al. (2019) SMErobotics: Smart robots for flexible manufacturing. In *IEEE Robotics & Automation Magazine* 26(1): 78–90.

Schmidbauer C, Komenda T, Schlund S (2020) Teaching cobots in learning factories - User and usability-driven implications. *Procedia Manufacturing* 45(Apr): 398–404.

Siemens Product Lifecycle Management Software Inc. (2011) *Alternate assemblies*. Retrieved from: http://support.industrysoftware.automation. siemens.com/training/se /en/ST4/pdf/spse01685-s-1040_en.pdf. [Accessed 19 February 2020]

Tajima S, Wakamatsu S, Abe T, Tennomi M, Morita K, Ubata H, et al. (2020) Robust bin-picking system using tactile sensor. *Advanced Robotics* 34(7–8): 439–453.

Thomas U, Finkemeyer B, Kroger T, Wahl FM (2003) Error-tolerant execution of complex robot tasks based on skill primitives. In *Proceedings - IEEE International Conference on Robotics and Automation* 3, 3069–3075.

Thomaz AL, Breazeal C (2008) Teachable robots: understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence* 172(6–7): 716–737.

Transeth AA, Stepanov A, Linnerud ÅS, Ening K, Gjerstad T (2020) Competitive high variance, low volume manufacturing with robot manipulators. In *2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*, 1–7.

Tsarouchi P, Athanasatos A, Makris S, Chatzigeorgiou X, Chryssolouris G (2016) High level robot programming using body and hand gestures. *Procedia CIRP* 55(Dec): 1–5.

von Drigalski F, Schlette C, Rudorfer M, Correll N, Triyonoputro JC, Wan W, et al. (2020) Robots assembling machines: learning from the World Robot Summit 2018 Assembly Challenge. *Advanced Robotics* 34(7–8): 408–421.

Wang L, Gao R, Vánca J, Krüger J, Wang X, Makris S, et al. (2019a) Symbiotic human-robot collaborative assembly. *CIRP Annals* 68(2): 701–726.

Wang T, Li D, Liu X, Zhou X (2019b) Gesture control for human-robot interaction based on three-way decision model. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, 311–316.

Warfield B (2020, September 22) *CNCCookbook 2016 CAD survey results, part 1: market share - CNCCookbook: be a better CNC'er*. CNC Cookbook.

Watson J, Miller A, Correll N (2020) Autonomous industrial assembly using force, torque, and RGB-D sensing. *Advanced Robotics* 34(7–8): 546–559.