2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Auto-Assembly: a framework for automated robotic assembly directly from CAD.

Fedor Chervinskii[§], Sergei Zobov[§], Aleksandr Rybnikov[§], Danil Petrov[§], Komal Vendidandi[§]
ΛΓΓΙVΛL

***Abstract*—In this work, we propose a framework called Auto-Assembly for automated robotic assembly from design files and demonstrate a practical implementation on modular parts joined by fastening using a robotic cell consisting of two robots. We show the flexibility of the approach by testing it on different input designs. Auto-Assembly consists of several parts: design analysis, assembly sequence generation, bill-of-process (BOP) generation, conversion of the BOP to control code, path planning, simulation, and execution of the control code to assemble parts in the physical environment.**

***Index Terms*—industry 4.0, smart manufacturing, cyber-physical systems, smart factory, manufacturing automation, manipulators, cellular manufacturing, digital twins, robotic assembly**

I. Introduction

Assembly planning is one of the most laborious tasks when releasing a new product for manufacturing. Thus, many algorithms and methods around computer-aided design (CAD) and digital twins of the factories have emerged in recent years that help process engineers to prepare a new design for assembly (Computer-aided Assembly Process Planning techniques [1]). An emerging trend of Industry 4.0 [2] suggests that a digital, highly automated factory should be able to infer the process from the design. In practice, even for an automated factory, assembly planning has to be followed by an offline-programming of all the robots and devices to perform the assembly plan.

In recent years, additive manufacturing technology, also known as 3D printing [3], has demonstrated a high degree of process automation. With this technology, it is now possible to load a computer-aided design (CAD) file into a machine and obtain a part that matches the desired design. In this paper, we explore the feasibility of achieving a similar level of automation in assembly processes. Our study considers scenarios in which input parts are pre-processed and placed in specific input jigs, and investigates whether a robotic cell or factory can perform the required assembly steps in a sequence, inferred automatically from the design.

In this work, we show how this can be achieved under specific constraints, paving the road for future experiments towards a more general approach and wider applications. However, the framework we propose is general enough to accommodate more complex designs and conditions, like many types of tooling and different joining technologies.

[§]Equal contribution. Research conducted while working for Arrival. The authors are no longer affiliated with the company. Correspondence e-mails: fedor.chervinskii at gmail.com, sergey at szobov.ru, arrybn at gmail.com

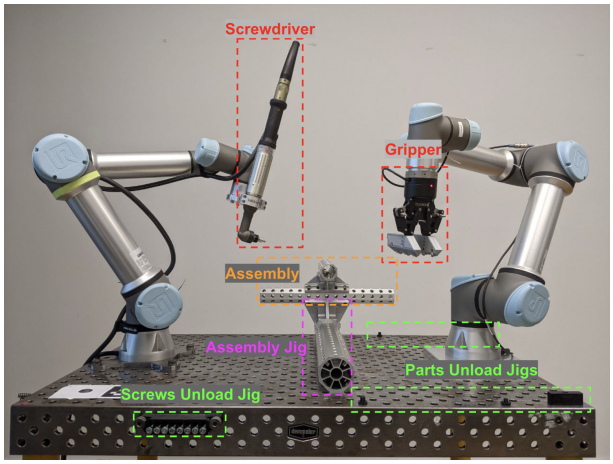

Fig. 1: Experimental setup: robotic cell with two UR5e manipulators. *Left*: UR5e with a screwdriver Likratec EH2 R1030-A and *Right*: UR5e with gripper Robotiq 2F-85 with custom designed gripper clamps. *On the table*: custom-designed 3D-printed jigs.

II. Related Work and Background

An Assembly Planning for a given design typically starts from identifying the mating features or joints and suggesting a feasible Assembly Sequence, which could be automated as seen in [4], [5], [6].

To proceed to the process planning, a virtual environment, also known as Digital Twin [7] is necessary. There are attempts to develop a common ontology, e.g. [8], [9] and unify interfaces between systems [10] to support process design automation.

Sierla, Seppo, et al. [11] discuss the conceptual framework of automated assembly planning using a digital twin. It uses the XML-based AutomationML [12] data modeling framework. This framework aggregates different data exchange formats like CAEX for plant description, COLLADA for geometry and kinematics of 3D models, etc.

There is still not sufficient work in joining together process planning, motion planning and execution using a common framework. In [13] authors used artificial intelligence to solve a Tooling Matching problem and developed an add-on for Octopuz [14] to do a Motion Planning and Robot Program Generation for disassembly, but not testing in physical cells. In another work, [15] a similar pipeline is described for an

architectural domain, mainly focusing on parametric design and modular assembly.

We claim that Auto-Assembly is the first proposed framework that can generate and execute robotic assembly process for an arbitrary input CAD design.

III. Problem Statement and Method Overview

The main objective of our work is to create a framework that enables a closed loop between design and robotic manufacturing. A target framework should analyse the design and provide a simulation of assembly, executable programs (when possible) and other feedback. The primary aim of the feedback is to help in adapting the design and manufacturing to better correspond to each other.

The feedback we should provide can be split into two categories:

- Successful simulation and its' artefacts can be directly used to decide on physical manufacturing. Users can choose between different processes to choose the one, based on the key performance indicators (KPI) they want to optimize: time, tooling price, energy consumption, etc.
- In case of a failure, the system should provide all necessary feedback that helps to change the design, robot's position, choose the robots with better parameters or different cell configuration. Such feedback can be: failed operations, missing appropriate tooling, parts or tools in collision, unreachable states.

To achieve this, we implement a framework described in detail in Section IV. Section IV-A gives an overview of our system and its components. Section IV-B discusses 3D modelling of the assembly design files that form the base of our data extraction pipeline. Section IV-C reviews the usage of this extracted data to produce a set of possible assembly sequences.

Each operation in the assembly sequence is enriched with tooling information as discussed in section IV-D. Finding a specific cell that contains all the resources like jigs, robots, and their tooling, etc to execute all the operations needed for an assembly is explained in section IV-E. Section IV-F tells about the generation of the control code that moves the robots to grasp, place and fasten parts in a cell.

In section V, we test our framework on different assemblies and discuss the results. In section VI, we review our findings from the experiments and future work.

IV. Framework

A. System architecture

Auto-Assembly framework can be divided into two parts as shown in Fig. 2. The first part, called "Artefacts generation", works with CAD files provided by a design engineer. It is intended to run once on the input data and provide artifacts, which can then be stored and re-used to run the assembly process in the simulated and physical environments. This part includes Assembly Sequence Generation, Tool and Cell Matching, Bill-of-Process (BOP) Generation and Control Code generation.

The second part can be seen as a deployed environment. It is represented as a system where we have many services, providing "abilities" which can be called from the domain-specific Process language (PL). An example of the PL script can be seen in Figure 3. Here we describe the most important services and their respective abilities:

- Robot Controller
  - Abilities to control the robots on a low level. As input, it takes a trajectory as a list of a robot's joint states, and as output, interpolates the trajectory and moves the robot.
  - Abilities to control tooling connected to the robot, like grippers, screwdrivers, etc.
- Motion Planner
  - Ability to plan a trajectory in the cell to move a robot to a target pose with other cell objects considered as obstacles.
- Jig Controller
  - Ability to return a pose of a part in a jig with respect to the jig origin.
- Assembly Service
  - Ability to retrieve the information about fasteners and resulting parts' pose with respect to the cell origin.
- Transform Service
  - Ability to get the position of any object inside a cell with respect to any object in the cell.
- 3D Simulator
  - Abilities to load objects from cell description and visualize cell state.
- Database and Message Bus
  - Abilities to publish and retrieve JSON objects. This component is used as a message bus and data storage.

All system parts exchange the data in a special format called Factory Control Model (FCM). It can be considered as a schema and also is a vital part of our system since it lets all the components speak the same language.

B. CAD Data Preparation and Extraction

For any given assembly, our framework needs two design files.

- Design file containing part assembly with joints. Fasteners are labelled as separate joints in order to distinguish them from other parts.
- Design file containing the jigs and gripper at different stages of assembly like grasping, placing, etc.

The examples of these files for an assembly are depicted in Figs 6 and 7 and are created by us in Fusion 360. Our method is CAD-software agnostic as long as we can extract the CAD data using an API.

From the design file in Fig. 6, we extract the joints and part occurrences information using Fusion 360 API [16]. Using this data, a joint register is created that maps every joint to its parts. The joint register follows the FCM schema.

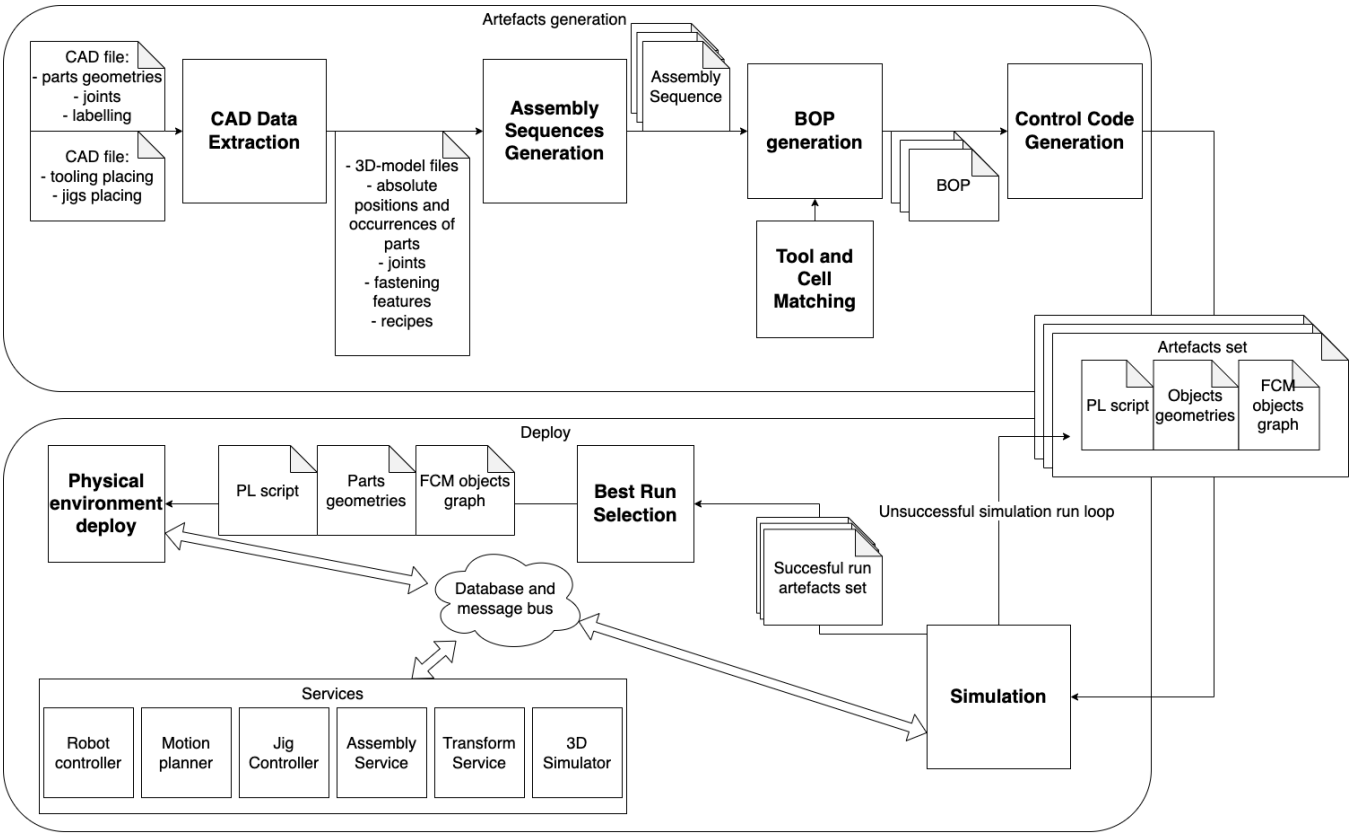

Fig. 2: System architecture

```
move_robot_to_position(cell, motion_group, manipulator_service,
                       planning_object, position, move_type,
                       ignored_collisions, ignored_collision_pairs) {
  rules {
    ~ get_cell_state(cell = cell, out cell_state = cell_state)
    ~ plan_trajectory(
        position = position,
        motion_group = motion_group,
        planning_object = planning_object,
        planning_socket_name = "eef",
        move_type = move_type,
        ignored_collision_pairs = ignored_collision_pairs,
        ignored_collisions = ignored_collisions,
        cell_state = cell_state,
        out result_motion_plan = trajectory
    )
    ~ execute_trajectory(trajectory = trajectory)
  } seq
  constraints {
    ~ execute_trajectory.@provider_id.resource_id == @manipulator_service.id
  }
};
```

Fig. 3: Listing of PL-code implementing high-level of robot control. Abilities *get_cell_state* and *plan_trajectory* are implemented by Motion Planner and *execute_trajectory* by Robot Controller

From the design file in Fig. 7, we extract the pose of gripper occurrence relative to the part during grasping it from the jig and placing it at the assembly state using the Fusion 360 API. We call this data as **recipes**.

C. Assembly Sequence Generation

A CAD design contains a lot of important information about the part's geometries, relations, and absolute poses. But what it lacks - the right assembling order - is the key information to move towards the assembled product. Assembly sequence encodes the order of operations needed to be performed on parts by the robotic cell. Although the operations can be executed sequentially, assembly sequences are represented by polytree (directed acyclic graph whose underlying undirected graph is a tree). Not any such tree represents a valid and feasible assembly sequence:

- only directly joined parts should be neighbours;
- the order of operations should take into account the geometrical limitations;
- the number of generated assembly sequences should be reasonably limited. Naturally, it grows exponentially with the number of parts involved. This makes it hard to check all the generated sequences to pick the best one according to some criteria.

The assembly sequence generation step aims to solve all three aforementioned issues, providing a limited number of valid sequences. The whole process can be divided into three steps:

- a liaison graph generation;
- assembly sequences generation based on the obtained liaison graph;
- geometry feasibility checking based on parts geometries

This approach we used is described in [5]. Further, the high-level steps, important implementation details, and differences with the original paper are described.

1) Liaison graph generation: The CAD file consists of the individual parts combined together with joints and fasteners. The information about the joints is crucial to accurately determine parts connectivity. Considering the parts as liaison graph nodes, connectivity information transfers into edges in this graph. We extracted the information about joints and fasteners from the design in CAD software to build up a liaison graph to further analyze it and generate assembly sequences.

2) Sequences generation: An assembly sequence determines the order of operations on parts. The liaison graph itself, being undirected, doesn't set the order of operations in general. But the order should be based on the liaison graph since the latter contains the information about the connectivity in the resulting assembly. Usually, there are many sequences of operations. [5] describes the approach of extracting all possible assembly sequences from the liaison graph. We followed the suggested approach.

3) Geometry feasibility checking: The geometrical feasibility of an assembly process is the fundamental property, which should be checked first to eliminate irrelevant sequences. These irrelevant sequences could contain, for example, one part to be joined with another part, which is trapped already inside the sub-assembly. To prevent this, geometrical analysis of sub-assemblies is used. One sub-assembly is translated step-by-step w.r.t another sub-assembly in one of chosen directions until the bounding boxes of the sub-assemblies still intersect and the solid bodies' intersection is checked. If the intersection represents a volume, it's impossible to join the sub-assemblies in the chosen direction, and the remaining directions should be checked.

Choosing the directions of translations alongside step size is important for the result. Due to the nature of assembly parts and their orientation alignment, directions along the main coordinate axes work well in the tested assemblies. In other cases, information from joints from the CAD file could be used to determine the potential directions. Step size is computed based on the minimal size of the part across both sub-assemblies. Precisely, the step size is computed as a 0.75 ratio of the diagonal of the smallest bounding box part. The idea behind this value is to exclude the possibility of going completely through the smallest part with a single translation step.

D. Tooling Matching

The assembly sequence in itself doesn't require specific tooling models, but this information is required for the next steps in the assembly process. Given a graph of the assembly sequence from the previous section, we traverse this graph, considering the type of operation and parts used, assigning all the tooling models and adding recipes to process this operation.

To archive this, we extract the following information from the CAD files:

- For grippers:
  - Model of the part gripper can be applied.
  - List of positions for grasping the part, calculated with respect to the part origin. We use the information from "joints", such as JointAxis, to extract the vector of connection. Based on this vector poses are calculated.
  - States of digital inputs register to control the gripper.
- For jigs:
  - Model of the part jig can hold.
  - Position of a part in a jig.
- For screwdrivers:
  - Screw-picking requirements, such as type of screw-holder.

We store this data in Tooling Database. In our approach Tooling Database is a storage with an API which allows adding, matching and visualizing of the tooling.

By analyzing the dataset of the tooling used in physical world production in the automotive field, we concluded that the same information is stored in the tooling design files and propagated to the tooling integration in the physical cells. We decided to formalize the requirements and then store this data. For the cases where it can't be calculated from the design files, we can manually put that information into the tooling database.

E. Cell matching

Cell description includes all the information representing an assembly cell. Cell description is used to deploy both environments (virtual and physical) and to choose a cell to execute Assembly Sequence. We topologically sort a graph of the Assembly Sequence and assign a level for every operation. The level is required to assign resources for the parallel operations when we should use different resources of the same model. Then by traversing each operation, we check the resources' models required for this operation and find their representation in the cell. If there are no cells satisfying all the resource requirements for the Assembly Sequence, we fail, providing feedback with the exact operation and the resources model we were not able to assign. As a result of the execution of the described algorithm, we have an assembly sequence to be converted into a BOP.

F. Control code generation

We match each operation in the Bill of Process (BOP) with a self-contained PL-script that performs the required operation, and pass the operation's resources and parts as parameters to create a single PL-script for assembling the product. An example of PL-script implementing unload operation is presented on Figure 4.

V. Experiments and Results

The objectives of our experiments are:

- To evaluate the framework.
- To evaluate the assembly BOPs in the physical environment to provide metrics and feedback on the assembly.

```
unload_operation(operation, cell, jig, out part_instance_id){
rules {
    ~ read_fcm#find_part(
        object_id = @operation,
        level = 2,
        format = "array",
        object_filters=[{"path": "type", "operation": "EQ", "value": "part"}],
        link_filters=[{"path": "type", "operation": "EQ", "value": "uses"}],
        out id = part)
    ~ get_part_pose_wrt_jig(
        jig_id=@jig.id,
        abs_occurrence_id=@part.abs_occurrence_object,
        out pose_from_jig)
    ~ unload_part(
        part_object = @part,
        cell_object = @cell,
        pose = @pose_from_jig,
        out part_instance_id)
    } seq
};
```

Fig. 4: Listing of PL-code for unload operation

The assembly we chose to test is shown in Fig. 6 and its tooling, and jig design are shown in Fig.7.

A. Data Preparation

- We extract the joint register and recipes as mentioned in section IV-B.
- Given the joint register and part models files, the Assembly Sequence Generator produced 8 assembly sequences for this assembly. These sequences are mentioned in Fig. 5.
- We enrich the assembly sequences using tool matching mentioned in section IV-D.
- We convert the enriched assembly sequences to BOP using cell matching as mentioned in section IV-E.

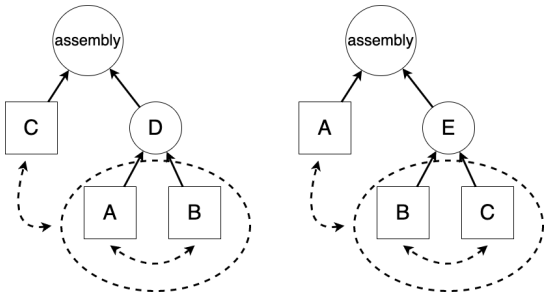

Fig. 5: A diagram of assembly sequences generation process for the design used in the experiments. Square blocks represent parts while circles represent (sub-)assemblies(D and E). The possible sequences are *Left*: ABDC, BADC, CABD, CBAD and *Right*: BCEA, CBEA, ABCE, ACBE.

B. Scene Preparation

: Before starting the assembly, if it's a simulated environment, the jigs are unloaded at the same poses as in the physical world in Fig. 1. If it's the physical environment, the jigs and parts are placed in their respective poses.

C. Simulation

1) Start simulation deployment with services and a database and message bus instance as mentioned in Fig. 2.

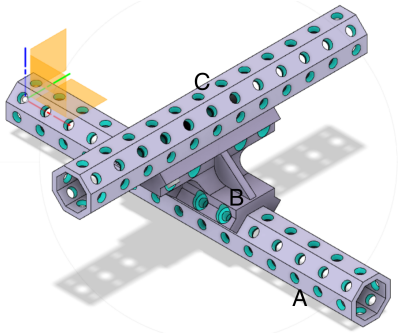

Fig. 6: A simple assembly containing 3 parts. *Profiles*: A, C and *connector*: B

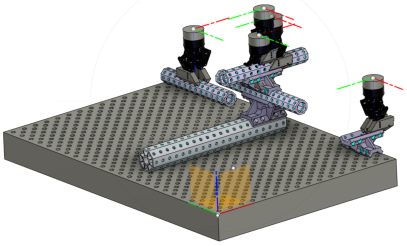

Fig. 7: Assembly Design file showing the assembly jig, custom-designed gripper adapters, grasping, and insertion states of the gripper. *Assembly state*: Center of the table. *Jig state*: Top and bottom right of the table.

2) We trigger execution of the PL code, which starts from running operations of type "unload" on input parts. This operation effectively initializes part instances on respective positions in the input jigs, so that the parts are now represented in the digital twin of the cell, as active objects with poses, visible for the simulator as well as for the motion planner.
3) The rest of the PL code is executed, sequentially reading necessary gripper positions, planning and executing trajectories, and triggering gripper control programs for grasping/releasing/fastening, all by calling respective PL functions that use abilities of underlying systems described in IV-A.
4) The user can observe the execution of assembly in the 3D simulator. During the execution of the assembly process, the motion planner gives us direct feedback, on whether it can reach a certain pose in the assembly or not.

For our example assembly, the initial results were the following:

- Of all the possible 8 assembly sequences, only one assembly sequence (ABDC) passed through the cell matching, as the cell resource descriptions (in this case jigs) support this. Many sequences get filtered based on the cell resources (jigs, robot tooling, etc.).
  All the sequences for any given assembly are feasible, if the cell has the resources to hold sub-assemblies, For example, the assembly sequence CABD is possible

when the cell has the jig that supports moving part C first to the assembly pose, then creating a sub-assembly D by moving parts A and B in the same order. Now the question here is, how to make the decision on which resources in this case jigs are needed to be designed to hold the sub-assemblies. If there are cycles in the graph, multiple BOPs pass through the cell matching which needs the same cell resources, which enables us to simulate and select the best one based on the metrics.

- We also noticed that our initial design failed due to fastening robot reachability, we took this feedback from the framework and changed the fastening position in the assembly to assemble a product.

To adjust the design, we changed the positions of the screws in the assembly to other holes without losing the structure stability. After this design adjustment, we were able to successfully simulate the one feasible assembly sequence.

This is one of the main features of the proposed framework - to get this kind of feedback about the product/tooling/cell design compatibility as soon as possible with minimal manual input.

D. Running assembly on physical robotic cell

Once we find an assembly sequence that passes in the simulation, we can proceed to the physical assembly process.

This is achieved by running the same generated PL code as before but now in a physical robotic environment. The only thing that differs compared to the previous pipeline in simulation is the first step - deployment of the systems. For a physical assembly we deploy the robot and the gripper drivers to be connected to robots and devices, such that in parallel with updating the state of the digital twin, these controllers will be changing the states of the tools in the physical world, such as robots moving along precomputed trajectories, gripper opening/closing and the screwdriver fastening the screws.

We evaluated this assumption on the physical robotic cell with two collaborative robots the layout of which can be seen in Fig. 1. The results of these experiments are two folds:

1) On one side, we can see that as soon as the digital twin is accurate enough, all computed gripper positions allow performing most of the operations, such as picking a screw, grasping and releasing a part, and in some cases to fasten a screw.
2) On another side, some operations show that an accumulated tolerance stack of robot calibration, tool accuracy, and parts accuracy leads to the inability to perform the joint operation such as fastening successfully, and the screwing position requires correction.

The example of running assembly in the virtual and physical environments can be seen in Fig. 8. The process of assembly of the provided CAD by running the generated PL code can be seen in the accompanying video.

VI. Conclusion and Future Scope

In this paper, we implemented and tested a framework to run a robotic assembly of a product by using only CAD files as input. We were able to use the feedback provided by the framework to change the original design and achieve successful assembly. We re-iterated the whole pipeline and transferred the assembly from the virtual to the physical world. We conclude that this transfer can be done only if the digital twin matches the physical cell precisely, which requires additional work, such as robots and cell calibration, but it's out of the scope of this paper. The choice of design of our system proved its flexibility since we were able to analyze and change artefacts produced during the different steps of the execution. The system is general enough to support new products and cell configurations.

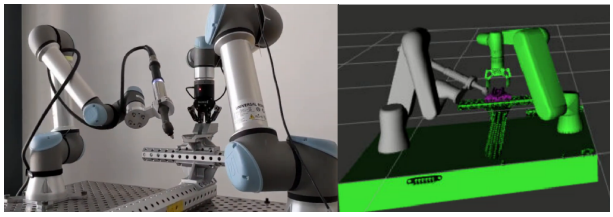

Fig. 8: Assembly process. *Left:* Physical environment and *Right:* Virtual environment.

The next step is to validate our framework on more complex assemblies, including new types of operations and operations which involve more than two parts.

In our experiment, we relied only on the parts' dimensional precision and the accuracy of the robots. While it could work for some parts, and partially worked in our case, it would likely fail on many other parts and materials. To address this problem, computer vision and other perception methods should be introduced into the framework to deal with variations in the real assembly process.

In the section IV-C3 the constraint we chose could lead to some possible assembly sequences being rejected. To solve this issue, we plan to implement a geometrical feasibility check based on joints from the CAD files or other optimization algorithms.

The method used in the section IV-E can lead to a sub-optimal configuration or even to setups where some robots can't reach parts. This approach was chosen as the easiest to track and implement. In future works, we plan to implement a more sophisticated scheduling algorithm based on geometrical and utilization constraints.

References

[1] Henrioud, J. M., and A. Bourjault. "Computer aided assembly process planning." Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 206.1 (1992): 61-66.

[2] Thoben, Klaus-Dieter, Stefan Wiesner, and Thorsten Wuest. ""Industrie 4.0" and smart manufacturing-a review of research issues and application examples." International journal of automation technology 11.1 (2017): 4-16.

[3] Masters "Computer automated manufacturing process and system" U.S. Patent 4,665,492. (1984)

[4] Deepak, B. B. V. L., et al. "Assembly sequence planning using soft computing methods: a review." Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering 233.3 (2019): 653-683.

[5] Vigano‘, Roberto, et al. ”Assembly planning with automated retrieval of assembly sequences from CAD model information” DOI 10.1108/01445151211262410
[6] Zhang, Jie, et al. ”Automatic assembly simulation of product in virtual environment based on interaction feature pair.” Journal of Intelligent Manufacturing 29.6 (2018): 1235-1256.
[7] Grieves, Michael. ”Digital twin: manufacturing excellence through virtual factory replication.” White paper 1.2014 (2014): 1-7.
[8] Järvenpää, Eeva, et al. ”The development of an ontology for describing the capabilities of manufacturing resources.” Journal of Intelligent Manufacturing 30.2 (2019): 959-978.
[9] Lemaignan, Severin, et al. ”MASON: A proposal for an ontology of manufacturing domain.” IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS’06). IEEE, 2006.
[10] Rust, Romana, et al. ”COMPAS FAB: Robotic fabrication package for the COMPAS Framework” Gramazio Kohler Research, ETH Zurich. https://github.com/compas-dev/compas_fab (2018)
[11] Sierla, Seppo, et al. ”Automatic assembly planning based on digital product descriptions.” Computers in Industry 97 (2018): 34-46
[12] Drath, Rainer, ed. AutomationML: the industrial cookbook. Walter de Gruyter GmbH & Co KG, 2021.
[13] Beck, Joshua, Alexander Neb, and Katharina Barbu. ”Towards a CAD-based Automated Robot Offline-Programming Approach for Disassembly.” Procedia CIRP 104 (2021): 1280-1285.
[14] Octopuz® https://octopuz.com
[15] Huang, Yijiang. Automated motion planning for robotic assembly of discrete architectural structures. Diss. Massachusetts Institute of Technology, 2018.
[16] Autodesk® Fusion 360® https://www.autodesk.com/