Pattern Recognition for Knowledge Transfer in Robotic Assembly Sequence Planning

Ismael Rodríguez[1,2], Korbinian Nottensteiner[1], Daniel Leidner[1], Maximilian Durner[1], Freek Stulp[1] and Alin Albu-Schäffer[1,2]

***Abstract*—The autonomous assembly of customized products is highly demanded in future manufacturing scenarios. This requires robotic systems being able to adapt to individual products without increasing overall production time. However, increasingly complex assemblies lead to a growing number of potential assembly sequences that have to be considered. To cope with this, we present an algorithm that is able to transfer previously identified assembly constraints to novel product variants. This reduces the search space, and thus planning times. The approach consist of three main steps. 1) Deduct semantic assembly constraints, from an analysis of feasible and unfeasible solutions. 2) Match key features of assemblies on a semantic level, by performing graph matching in the representation of the assemblies. 3) Use pattern recognition and classification based on machine learning techniques to transfer the knowledge of constraints for sub-assemblies into the complete assembly. We demonstrate our contributions on a two-armed robotic setup that assembles product variants out of aluminum profiles.**

***Index Terms*—Assembly, Intelligent and Flexible Manufacturing**

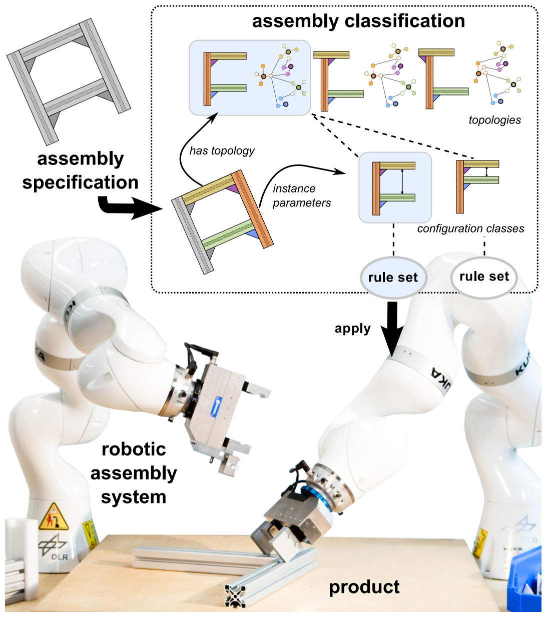

Fig. 1. Flow of the overall system. It begins by introducing an assembly specification. This description is compared with previously learned topologies via a graph matching. Once a match is found the instance parameters are extracted, these parameters determine via a classification algorithm which rule set should be applied. Given the rule set, the robotic assembly system is programmed.

I. Introduction

The shift from a product-centered perspective (mass production of one article) to a customer-centered perspective (mass customization of product variants) dramatically changes the requirements for assembly processes. In mass production, human engineers adapt production lines for novel product variants. But if each product becomes a variant, this is no longer feasible and the adaptation must be automated. This requires robots and workcells that are flexible enough to assemble many product variants and algorithms which automate the planning of the assembly accordingly.

However, automation and flexibility also come at a cost. In order to achieve both characteristics simultaneously several time consuming checks have to be performed. As an example [1], assembling an IKEA chair required 11 minutes of motion planning alone, for a fixed sequence of assembly steps. In the case of product variants, the assembly sequence space itself must be explored, whereby the search may easily take hours or days instead of minutes.

To make assembly planning more efficient and general, we exploit the fact that a palette of related products is typically built out of a shared catalog of common components. Therefore, reusing previous knowledge acquired for assembling individual products has the potential to speed up future planning times for the entire product palette. In particular, we propose an algorithm (KT-RASP Knowledge Transfer - Robotic Assembly Sequence Planner) that identifies recurring sub-assemblies, and re-uses previously generated assembly constraints for these sub-assemblies to speed up future assembly planning (Fig. 1). This approach raises two main research questions that are investigated in this work: How do we represent assemblies such that we are able identify common sub-assemblies? And how do we transfer the knowledge acquired from each instance of the identified sub-assemblies to further cases?

Given these questions, the main contributions of this work are: (i) a graph representation that is able to describe (sub)-

Manuscript received: September, 9, 2019; Revised November, 15, 2019; Accepted January, 21, 2020.

This paper was recommended for publication by Editor Dan Popa upon evaluation of the Associate Editor and Reviewers' comments. This work is supported by the Deutscher Akademischer Austauschdienst (DAAD).

[1]Ismael Rodriguez, Korbinian Nottensteiner, Daniel Leidner, Maximilian Durner, Freek Stulp and Alin Albu-Albu-Schäffer are with Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Wessling, Germany. ismael.rodriguezbrena@dlr.de

[2]Ismael Rodriguez and Alin Albu-Albu-Schäffer are with TUM - Technische Universität München, Faculty of Informatics, Garching, Germany.

Digital Object Identifier (DOI): see top of this page.

assemblies by means of semantic relations, (ii) a pattern matching algorithm that is able to identify known sub-assemblies represented by means of graphs, (iii) a novel concept to automatically generate assembly sequence rules, and (iv) suitable machine learning strategies to assess if these rules can be transferred between sub-assemblies.

This work builds on our previous work [2], which introduced the assembly planning algorithm, but did not consider knowledge transfer between different assemblies to speed up planning. To make this article self-contained, we have summarized relevant parts of [2] in Sections III-A1, III-A3, and III-D. All other sections describe the novel contributions listed above.

The remainder of the paper is structured as follows. In Section II we describe the related work. We introduce the different representations needed to describe our concept in Section III. The training process of our system and how it is utilized is described in IV and V respectively. Finally, we evaluate our results in VI and eventually end with a discussion in Section VII and conclusions in Section VIII.

II. Related Work

We present related work in three different areas: assembly sequence planning (the application), knowledge transfer in assembly planning (the specific topic addressed in this paper), and machine learning in graphs (which is key to our concept).

A. Assembly Sequence Planning (ASP)

One major issue of the assembly sequence planning problem is that it constitutes an *NP*-hard combinatorial problem, since in principle the amount of orderings and therefore possible solutions grows as a factorial of the parts involved [3]. As a historical reference we recall the FLAPS system [4], which uses semantic symbols to model the interference, connection and contact of the parts of an assembly. The FLAPS system shows one of the first attempts to reduce the amount of solutions that have to be analyzed by the assembly planner using the before mentioned semantic symbols. In more recent approaches, more information about the different restrictions of the system are taken into account. In [5], plans are generated based on geometrical and grasping constraints. Here, a *disassembly for assembly* strategy is combined with an integrated grasp planner to find feasible solutions. This strategy was successfully used in our robotic setup [6]. In [7], a more advanced algorithm is used to find an optimal assembly sequence, by penalizing or enhancing particular sub-assemblies or tasks.

In our previous work [2], we proposed to consider efficiently the restrictions arising from the robots kinematics, dynamics, as well as the utilized skill set, and not only the restrictions arising from the product itself. The developments conducted in this prior work are based on a semantic representation of geometric constraints in form of priority rules. This rule concept is reused in this work, as it is exploited during different stages of our learning process, where we transfer rule knowledge between different sub-assemblies.

B. Represent, Transfer and Learn Assembly Knowledge

In order to transfer knowledge, a representation of it is needed. Knowledge representation is normally done by ontologies. With dedicated ontologies it is possible to use them to solve assembly problems such as [8], [9], [10], [11]. Such ontologies are detailed representations of the whole environment. Nevertheless it takes a lot of modeling effort of the environment, while we focus more on the representation of the product. Perzylo et al. [12] exploit CAD models to generate an ontology. In [13] the same authors include a concept of modeling relations as link between two surfaces which is adapted in our proposed method. The resulting graph model of related objects is substantial for the assembly representation developed in this article.

Recently, knowledge transfer has become increasingly prominent in assembly processes. This increase is aligned with the need for fast and flexible adaptation, not only for the assembly planning but for all the stages of the process like tuning of tasks, stacking processes or reusability of skills. For example, Liu et al. [14] propose a machine learning based approach to build a stacked wall. The authors define the final assembly to be built using deep reinforcement learning. In [15], similar learning techniques are used to parameterize different robot skills, such that there is no need for human intervention during task specification, which saves time in the assembly process. All of the literature reviewed has shown the same pattern, learning or knowledge transfer are performed in structured environments where the optimization function is clearly specified. On the other hand, to the best of our knowledge, no work in the field of assembly planning has achieved transfer of knowledge from previously build products to new ones. The main problem of this is the difficulty to find a suitable similarity measure to compare assembly structures, which is one key component of the approach presented.

C. Machine Learning in Graphs

We define an assembly structure as a graph which is an intuitive representation of relational structures and provides required means to define similarity measures between different (sub-)assemblies. Graph matching is relevant for a variety of fields, including data mining, neuro-linguistic programming, and chemical compounds. It describes the search for an exact match of (sub-)graphs or as measure for structural similarity [16]. Two main approaches exist, namely graph kernels and graph neural networks (GNNs). The former computes a pair of graphs based on substructures which can be sub-tree graphs [17], cyclic patterns [18], [19], or random walks [20]. GNNs on the other hand learn a graph representation by recursively aggregating local structural information [16], [21]. The produced embedding in a vector space is typically used for efficient similarity reasoning. Both approaches can also be applied to graph classification. While graph kernels hereby act in a two-stage manner – first extracting of implicitly defined features, second classifying them – GNNs are able to directly learn features which are meaningful for down-stream tasks such as classification [22], [23]. In this work we apply a two-stage approach by extracting features from the graph and forward them to a classifier.

III. Formal Assembly Representation

In order to transfer knowledge between different assemblies and sub-assemblies, a formal assembly representation is needed. In this section the concepts of *parts and surfaces* as elements composing an assembly are presented. The description of a *relation* between these elements is given, followed by the definition of *rules* as a semantic representation of constraints. Finally, we define the concept of *topologies* and how this is related to assemblies via the *instances of a topology*.

A. Parts and Surfaces

We describe assemblies as a set of parts. Parts are composed by a set of surfaces.

1) Parts: An assembly is composed by a finite set $\mathcal{P}$ of N parts: $\mathcal{P} = \{p_1, p_2, ..., p_N\}$.

Where each part p_i is specified by a name, a type, and its pose relative to the assembly.

2) Surfaces: Each part p_i is composed by a set of (a finite amount K_i of) surfaces $\mathcal{S}_i$: $\mathcal{S}_i = \{s^i_1, s^i_2, ..., s^i_{K_i}\}$.

These surfaces are specified by a type and a unique id.

3) Semantic Relations: Surfaces may be in a relation to each other (e. g. being in touch, screwed, inserted) which are represented semantically. A relation is defined by two surfaces and a label l indicating the kind of relationship:

$$\mathcal{R} : E(s^i_a, s^j_b, l),\ with\ s^i_a \in \mathcal{S}_i, s^j_b \in \mathcal{S}_j.$$

Using geometric analysis or prior knowledge, it is possible to deduct these relations by only knowing the description of the assembly (poses and types of parts).

B. Topology Representation

We define a topological representation by semantically describing the different parts of the assembly and their surfaces, i. e. long faces and short faces of profiles and two connecting faces of angle brackets. From this we construct a graph as shown in Fig. 2. The encoding of an assembly into a graph structure is done by only knowing the geometrical characteristics of each type of part and their relations. This representation is aligned with the approach before mentioned [13] where products are represented as a graph connecting different related surfaces.

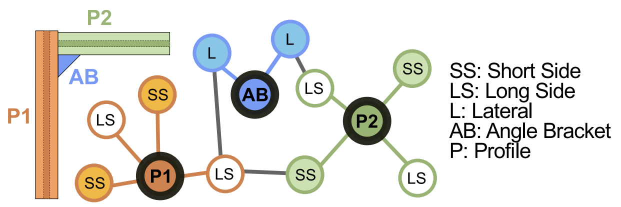

Fig. 2. Topology of an exemplary assembly. Each part and surface is represented as a node. The different gray edges represent relations between surfaces, the color edges indicate to which part each surface belongs. The profiles representation have two different types of surfaces, representing long and short sides.

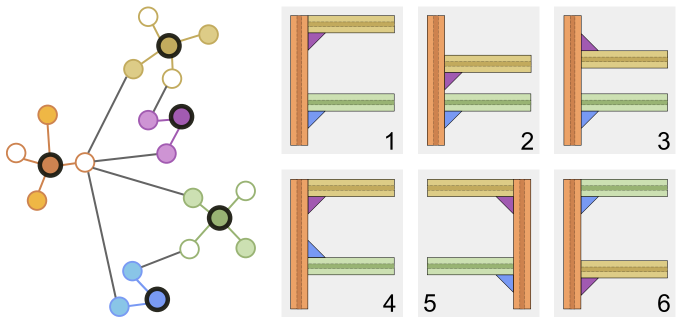

Fig. 3. Collection of assembly instances which are all represented by the same topology shown on the left. Having the same topology representation does not mean that the constraints to build each of these assemblies are the same. Based on the parameters of the instance we are able to classify which kind of constraints a particular instance has. For example instance number 2 is more constrained than instance number 1.

The topology of an assembly is formalized as a graph T composed by a set of nodes V and a set of edges E as seen in Fig. 2. The nodes in V represent the parts of the assembly as well as the surfaces of each part. The edges represent a relation between these nodes. An edge $e \in E$ may be between a part and a surface which is composed of, or between two surfaces which are related as previously explained.

Even though a clear mapping from an assembly to its representative topology is straightforward, it is not the same in the opposite direction. This is due to the fact that our representation of relation between surfaces does only encode a symbolic connection and not the geometric parameters. Accordingly, several assemblies may have the same topology. We refer to each of these assemblies with the same topology as an *instance* of a topology.

C. Instances of a Topology

Instances of a topology are any possible configuration of parts which match the relations indicated by the topology. Fig. 3 provides examples of different instances[1].

Given the big variability of these instances, it is crucial to understand that not all of them share the same constraints as they are assembled. For example, assembly number 2 has an extra constraint, i. e. one has to set the purple angle before the green profile. In order to identify such different cases, it is mandatory to know the relative positions between the different parts of an instance. A set of *parameters* is required to describe this relative positions. Here, we use the translation vector of the relative transformation between surfaces of related parts. In our particular case, since the surfaces are either perpendicular or parallel, we simply use the distance between them. This reduces the parameter vector, which is beneficial for classification.

A well-defined order of these parameters is required. This is achieved as we define particular instances of a given topology. All the instances have the same topological representation T, therefore a mapping between the parts of these instances and the parts of the topology T (nodes corresponding to parts) is possible. An example is given w. r. t. the assembly

[1]The figures are best appreciated in a color version.

in the Fig. 3, where each of the six instances are mapped according to the color code given by the topology, i. e. the orange profile is mapped to the orange node and so forth. The parameters are always build in the same order, e. g. the relative distances of the surface of the orange profile and yellow profile, then orange profile and purple angle bracket and so on. The procedure is repeated for every new assembly. As a result, a strict order of parameters is maintained which allows us to compare individual instances of a topology. These parameters together with the topology describe in the underlying domain the geometry of an assembly, since the distance between all pairs of surfaces allows to define the relative position between parts. Therefore an ordered array of parameters is uniquely generated from each assembly and thus representative for this particular instance. During the remainder of the paper, we will refer to an instance, whenever we talk about a particular assembly with a unique parameter set.

D. Rules

Prior constraint knowledge (i. e. whether a certain part prohibits the completion of the assembly) is transferred between instances by means of explicit representation of semantic rules. We introduced the concept of rules as a way to represent the different types of constraints in an assembly in our previous work. For the sake of completeness, a short introduction of the rules is given here. A detailed explanation is found in [2].

In a nutshell, we define *precedence* between parts. This precedence represents an ordered pair of parts. It indicates that p_x should be assembled before p_y, forming a partial order in the set $\mathcal{P}$:

$$AtomicPrecedence : AP(p_x, p_y),\ with\, p_x, p_y \in \mathcal{P}.$$

We show that it is possible to derive these rules based on human knowledge, by mapping the different constraints into certain rules. In order to diminish the need of human knowledge, a novel algorithm to deduct these precedence rules automatically is presented in this work. In summary, rules describe a formal representation of constraints. In this work, we use rules to transfer knowledge between different instances of a particular topology to avoid re-planning.

E. Clustering of Instances

Now after introducing the rules as a semantic representation of constraints, we use this idea to cluster the different instances. This clustering is useful since the knowledge we transfer are the constraints expected to appear in a novel instance, therefore we need to easily identify all instances with the same constraints. Each instance is mapped to a set of rules which applies to it. Therefore an array of parameters given by an instance is directly mapped to a set of rules. Since several instances can have the same set of rules, we cluster all these instances together in a particular configuration class. Therefore, for each array of parameters X a configuration class y is assigned, which enables to train a classification algorithm based on the generated data. This is explained in more detail in the following section.

IV. Training of the System

In this section we describe how the introduced representations interact to learn whether a constraint associated with a certain topology applies for a particular instance. First the algorithm which automatically deducts rules (semantic representation of constraints) is shown, followed by a description of the system's training process including the decision criteria to store topologies. Finally, we explain how the classification algorithm is applied.

A. Rule Deduction Algorithm

In our previous work, semantic rules were directly mapped from geometric constraints, where human expertise was used to define the initial mapping. Since our goal is to automate the whole assembly procedure without any human intervention, we aim for an automatic rule generation. In this work, we analyze all of the semantic solutions (i. e. the possible ordering of parts) and check whether each one is feasible or not. It is important to point out, that the different checks are performed in several *complexity levels* (i. e. different abstractions of the assembly system). These levels are represented by boxes in Fig. 4, showing the successive checks. The abstractions may vary in the physical fidelity (e. g. a static analysis of the geometries, using kinematics, or even dynamics), and also vary in the scope (e. g. considering only the parts, also the tools or the whole setup). The scope is actually most crucial, as the deducted rules do not only depend on the structure of the assembly, but also on the capabilities of the robotic systems. For a detailed description on the concept of feasibility checks w. r. t. physical fidelity and scopes please refer to [2]. In this work, it is relevant to know that, *the transfer of knowledge between assemblies, is not restricted to geometrical constraints, instead the whole information about the capability of the system to build this assembly is implicitly (via the rules) taken into account.*

In order to understand the algorithm of automatic rule generation, it is important to describe how the search of solutions (different orderings of parts) is conducted in our previous work. A Deep First Search algorithm is used to build the possible orders of the parts, as it is visualized in Fig. 4. For each branch of the search tree a possible solution is explored by adding new parts at each node until the whole assembly is completed. Two branches may share S nodes, meaning that the solutions represented by these two branches have the same sequence of S parts at theirs beginning.

One of the novelties of this paper is the rule deduction algorithm. First this algorithm searches for two consecutive solutions one feasible and the other not (marked with a yellow box in Fig. 4). By feasible we understand not only valid by following the existing rules, but also that this solution passes all checks in the different complexity levels. Once these two solutions are found, the algorithm looks for the lowest common ancestor between these two solutions (the yellow node in Fig. 4). Since under this node there are solutions which are feasible and some that are not, we deduct the parts already placed up to this node since they do not define if the solution will work or not. This property is useful to find which parts

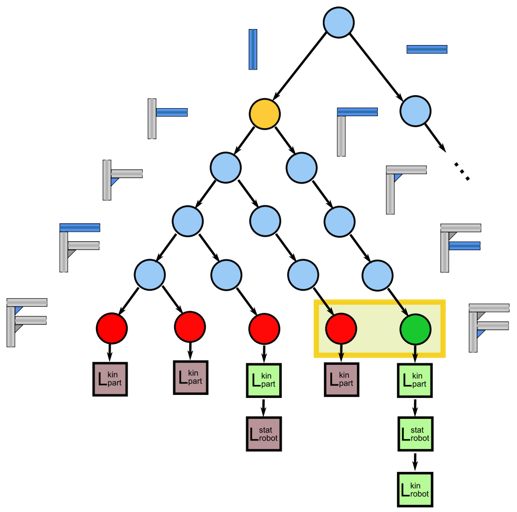

Fig. 4. Simplified DFS search tree for an F assembly. In yellow is marked the lowest common ancestor of the two consecutive solutions marked with the yellow box. Therefore the part which is set in the not working solution after the yellow node is key for the constraints found.

are critical to the constraints that we are looking for. That is, we know that the assembly raises no problems up to the state represented by the yellow node. The parts which are critical are the ones that are placed exactly after this node, especially the ones of failed (unfeasible) solutions (we call this part p_{fail}), since it has been placed too early. The rule then indicates that p_{fail} should be placed after a certain set of parts. To define this set of parts ($listOfReferents$), we compare p_{fail} against every other part p_{ref} and check if there is any successful solution where p_{ref} has been placed before p_{fail}. In the case that there is no successful solution with this property p_{ref} is added to $listOfReferents$. Then we generate all the atomic precedence rules

$$AP(p_{ref}, p_{fail}), \quad \forall p_{ref} \in listOfReferents$$

The time complexity of this algorithm is $O(TS + F \times N^2)$ where TS is the total amount of semantic solutions, F the amount of feasible solutions and N the amount of parts. In our previous work it was not necessary to explore all possible solutions, yet human knowledge was required to map constraints to rules. Accordingly, it cannot be used in any new domain without manual human mapping. In the new algorithm we assume that once there is a failure in a solution, adding more parts does not fix the issue, which holds for most industrial cases.

B. Generation of Databases

In order to use the system, we want to generate a knowledge base. In our case it is created by analyzing randomly generated assemblies. One by one the generated assemblies are evaluated. First, all semantic solutions are computed. Second, the rules that apply for this case are deducted. In Algorithm 2, we show the process through which the databases of topologies and instances are created.

The first decision to be made is which topologies should be stored in the database. Storing too few topologies may lead to a loss of relevant information, too many topologies make the system slow. A new topology should be stored if it *encloses* a rule that existing topologies do not. We say a topology *encloses* a rule if all the parts involved in this rule are present in such topology. If this is not the case, the topology representing the new assembly is stored as novel concept in the database.

Formally, we define $\mathcal{T}$ as the set containing all topologies t in the data base, $\mathcal{M}_t$ as the set of all matches m with the topology t. A match m is a bijective function $m : \mathcal{P}^t \rightarrow \mathcal{P}^{t_a}$ between the parts $\mathcal{P}^t$ of the topology t and a (sub-)set of parts $\mathcal{P}^{t_a}$ of the topology of the new assembly t_a. In order to find these matches we execute the function $getMatches(\mathcal{P}^t, \mathcal{P}^{t_a})$ (Algorithm 1), which looks for subgraphs in t_a with the same structure of parts and surfaces as defined by t. Therefore it assigns a part p^t_1 of t to one part $p^{t_a}_1$ of t_a having the same part type (line 6 in Algorithm 1). This process is repeated until all parts p^t_i are assigned. Next, the validity of the underlying graph match is evaluated. A match is considered valid if every part pair (p^t_1, p^t_2) has the same surface relation as their assignees in ($p^{t_a}_1$, $p^{t_a}_2$) (line 17 in Algorithm 1). Formally: $\mathcal{R}^{t_a}_{1-2} : E(s^{t_a,1}_a, s^{t_a,2}_b, l) = \mathcal{R}^t_{1-2} : E(s^{t,1}_{\hat{a}}, s^{t,2}_{\hat{b}}, \hat{l})$, where $\mathcal{R}^{t_a}_{1-2}$ defines the relation between parts $p^{t_a}_1$ and $p^{t_a}_2$ and $s^{t_a,1}_a$ one arbitrary surface of part $p^{t_a}_1$ (respectively for other parts and topologies). The relation between parts is considered to be equal when the surface pairs ($s^{t_a,1}_a$,$s^{t,1}_{\hat{a}}$) and ($s^{t_a,2}_b$,$s^{t,2}_{\hat{b}}$) are of the same type and both labels l and $\hat{l}$ are equal. If this does not hold, the algorithm recursively tries the next feasible mapping.

Algorithm 1: Get matches

```
1  $match \leftarrow \emptyset$, $validMatches \leftarrow \emptyset$
2  GET MATCHES ($\mathcal{P}^t, \mathcal{P}^{t_a}, match$)
3      if not $\mathcal{P}^t$ is empty then
4          for $p_t$ in $\mathcal{P}^t$ do
5              for $p_{t_a}$ in $\mathcal{P}^{t_a}$ do
6                  if sameType($p_t$,$p_{t_a}$) then
7                      remove $p_t$, $p_{t_a}$ from $\mathcal{P}^t$ and $\mathcal{P}^{t_a}$
8                      $match$.add([$p_t$,$p_{t_a}$])
9                      GET MATCHES ($\mathcal{P}^t, \mathcal{P}^{t_a}, match$)
10                     add $p_t$, $p_{t_a}$ back to $\mathcal{P}^t$ and $\mathcal{P}^{t_a}$
11                 end
12             end
13         end
14     else
15         for ([$p^t_1, p^{t_a}_1$] in match ) do
16             for ([$p^t_2, p^{t_a}_2$] in match ) do
17                 if not sameSurRel ($p^t_1, p^{t_a}_1, p^t_2, p^{t_a}_2$) then
18                     break GET MATCHES
19                 end
20             end
21         end
22         $validMatches$.add($match$)
23     end
```

We define P_r as the set of parts that are involved in a particular rule r e. g. if r is $AP(p_A, p_B)$ then $P_r = \{p_A, p_B\}$. Formally, we express that a rule r is enclosed by a topology t

if any of its matches ($m \in \mathcal{M}_t$) has as co-domain a set which contains the set P_r.

It is important to point out that the appearance of new topologies is affected by the rules learned. Since these rules are derived by the result of the checks in the different complexity levels, they depend on the robotic setup and its skills. The rules have an impact over which topologies are discovered as new. As a result, topologies also adapt to different robots' set of skills and setups.

Algorithm 2: Generation of databases

```
1  GENERATION OF DATABASES
     ($DBTopologies, DBInstances_t, newAssembly$)
2    $listOfSolutions \leftarrow$ generateSol($newAssembly$)
3    $listOfRules \leftarrow$ deductRules($listOfSolutions$)
4    $t_a \leftarrow$ generateTopology($newAssembly$)
      $needNewTopology \leftarrow$ False
5    for $r$ in $listOfRules$ do
6        for $t$ in $DBTopologies$ do
7            $\mathcal{M}_t \leftarrow$ getMatches($t, t_a$)
8            for $m$ in $\mathcal{M}_t$ do
9                if $m$ encloses $r$ then
10                   break for (goes to next rule)
11               end
12           end
13       end
14       $needNewTopology \leftarrow$ True
15   end
16   if $needNewTopology$ then
17       $DBTopologies$.add($t_a$)
18   else
19       for $t$ in $DBTopologies$ do
20           if parts(t)==parts($newAssembly$) then
21               $\mathcal{M}_t \leftarrow$ getMatches($t, t_a$)
22               for $m$ in $\mathcal{M}_t$ do
23                   $DBInstances_t$.add($m$)
24               end
25           end
26       end
27   end
```

In the particular case of only checking the geometries of the assembly at the training stage (in our testbed of only up to 3 profiles and 2 angle brackets) only three different topologies are discovered. These topologies (Fig. 5) are sufficient to enclose all the constraints of our testbed. Actually, even more topologies exist, e. g. the ones representing the assemblies of two profiles and one or two angle bracket. However, they are not stored in the database of topologies as they can be assembled in any sequence without constraints from our system. Therefore, they do not add any additional rule to the system. Hence, all solutions work unconstrained.

In the case no new topology is discovered and there is a match with an already existing topology with the same amount of parts as the assembly, we store a new instance of this topology. The instances are stored in different databases depending on which topology it comes from[2]. The instance stored is represented w. r. t. the parameters explained in Section III and the rules deducted for this particular instance which represent the constraints of the assembly.

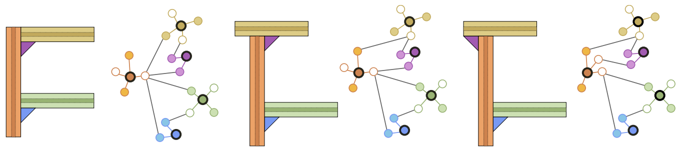
Fig. 5. Three topologies that the algorithm has found relevant for three profiles assemblies and only two angle brackets. Even though more topologies exist within assemblies of three profiles e. g. topologies with only two profiles, these are not relevant since new rules have not been learned with them.

C. Instances Classification Algorithm

The generated parameter array X is assigned with a configuration class label y related to the set of rules that were deducted. Once the assignment is done, we encounter a classical classification problem ($y = f(X)$; with f as mapping function), which can be solved by an arbitrary classifier. In this work we applied three different classifiers, namely Random Forest (RF), Support Vector Machine (SVM) and Neural Network (NN)[3]. To evaluate the suitability of the classifiers for the given problem, a 5-fold cross-validation for both scenarios was conducted. The accuracies averaged over the different topologies are depicted in Table I.

TABLE I
COMPARISON OF THE ACCURACY OF DIFFERENT CLASSIFIERS ON A 5-FOLD CROSS-VALIDATION

	SVM	RF	NN
Scenario A	0.69 ± 0.13	0.97 ± 0.03	**0.99 ± 0.01**
Scenario B	0.74 ± 0.05	0.87 ± 0.06	**0.92 ± 0.06**

As can be seen, in spite of the relatively small amount of training data the SVM and RF, both with optimized hyper-parameters, are (slightly) outperformed by the NN. A reason therefore is the complex feature space induced by symmetries of the profiles (certain features are switchable) which is not completely covered by the small amount of data. Hence, the classifiers have to be able to generalize, which the NN with dropout as regularization technique during training achieves more robustly. Note that the mentioned symmetries are expert knowledge and unknown for the system. Given the results, the following experiments are conducted with the NN.

V. PLANNING

Once the training is completed, the execution in production is straight forward (as shown in Algorithm 3). We match an unseen assembly to all topologies learned. An example of this is seen in Fig. 6.

Based on the identified matches, four particular instances with four different parameter sets are extracted. Once the

[2]A representation of instances stored is found in https://rmc.dlr.de/rm/en/staff/ismael.rodriguezbrena/assemblydata

[3]The network is composed of one input layer, two hidden layers (each 60 neurons), and one output layer. The activation function is a so-called Rectified Linear Unit (ReLU). During training the regularization technique Dropout (dropout rate 0.2) is applied to reduce the bias towards the data.

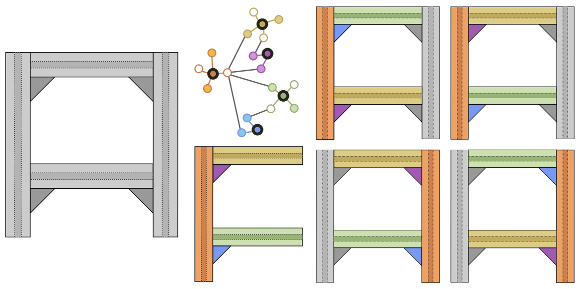

Fig. 6. All the different matches found for the assembly shown on the left . The assembly is composed of 4 instances of the F topology.

Algorithm 3: Execution with trained system.

```
1  EXECUTION ($DBTopologies, newAssembly$)
2      $t_a \leftarrow$ generateTopology($newAssembly$)
       $listOfRules \leftarrow \emptyset$
3      for $t$ in $DBTopologies$ do
4          $\mathcal{M}_t \leftarrow$ getMatches($t, t_a$)
5          for $m$ in $\mathcal{M}_t$ do
6              $x \leftarrow$getParametersOfMatch($m$)
7              $class \leftarrow$classifyWithRF($x$)
8              $listOfRules$.add(getRulesFromClass($class$))
9          end
10     end
11     buildSolutions($listOfRules$)
```

parameters are obtained, the trained classification algorithm indicates to which configuration class the instance should be classified. Given the class, we know the set of rules that should be used during assembly, these rules are used prune the search for solutions. We assume that if a sub-assembly has a certain set of constraints stand alone, at least the same constraints are present when it is part of bigger assemblies. If all the constraints were learned correctly, the solutions marked as semantically feasible should be executable by the real system.

VI. Evaluation

In order to demonstrate the capabilities of our system, we have designed two different evaluation scenarios. In the first scenario, we train our system to transfer knowledge on how to assemble aluminum profile structures by taking into consideration only a basic level of complexity. In this level of complexity we consider only checks for geometric constraints of only the parts of the product. Since these checks are simple, they can be done quickly. For this reason we trained our system with 5000 different assemblies of two and three aluminum profiles. We choose these basic ones since they enclose most of the existing constraints in the domain.

In the second scenario, we take into consideration more levels of complexity including checks involving the robot kinematics. Since data generation in this scenario is more costly, we train our system only with 1000 assemblies of two and three aluminum profiles. In this scenario, we limit the robots to an restricted set of skills which allows them to build a smaller variety of assemblies than in the previous scenario.

We train the system for each of these scenarios and we test them with 20 assemblies to verify that the knowledge

TABLE II
Results of the algorithm in two different scenarios. Showing the amount of solutions and the time consumed.

Case	Scenario A Only Geometry Checks			Scenario B Include Robots IK		
	Nr. Sol.	Time(ms) Old Sys.	Time(ms) KT-RASP	Nr. Sol.	Time(ms) Old Sys.	Time(ms) KT-RASP
1	0	183	250	0	280	256
2	16	249	202	8	573500	225
3	64	1202	424	0	71920	200
4	288	3829	1336	0	104360	232
5	576	5592	4256	96	>600000	674
6	32	315	388	16	>600000	161
7	16	146	166	8	>600000	216
8	32	1031	1896	8	160040	223
9	32	251	172	16	>600000	175
10	0	22	86	0	38	149
11	16	162	723	0	93410	147
12	32	242	1098	16	>600000	174
13	96	947	1485	0	106632	190
14	0	31	141	0	36	213
15	512	6448	2805	0	128675	279
16	3072	49723	20272	0	109484	378
17	96	991	669	0	103652	312
18	16	135	150	0	99568	152
19	64	621	363	0	96807	177
20	0	29	*	0	29	219

is correctly transferred from the basic sub-assembly to more complex cases. To validate the correct transfer of knowledge, we compare the amount of successful solutions generated by the system to the ground truth amount of feasible solutions. We measure the times needed for the old approach with different checks and the times required for our KT-RASP algorithm to find all feasible solutions. The results are shown in Table II.

In 39 out of the 40 analyzed cases[4] the solutions found by the KT-RASP algorithm were the correct ones matching the ground truth algorithm. The only case in which our algorithm returns false positives is the assembly number 20 of the first scenario which is shown in Fig. 7. The constraint of this case is related to the additional fourth profile, and it cannot be found in assemblies of only two and three profiles. Accordingly, it is not learned by the system.

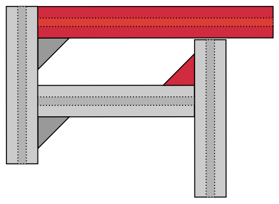

Fig. 7. Assembly with constraints that cannot be predicted based on data gathered with two and three profile assemblies. The constraint related to the two red parts is not known.

Regarding performance, the KT-RASP approach improves the planning times significantly. In the first scenario, the computation times are comparable. However, in some cases of the second scenario, the planning times are reduced from ten minutes to less than half a second. This shows that the KT-RASP algorithm is able to scale to more complex checks, which would otherwise significantly increase planning time.

[4]The test cases can be found in https://rmc.dlr.de/rm/en/staff/ismael.rodriguezbrena/testcases

VII. Discussion

The planning times are improved with KT-RASP in some cases up to several orders of magnitude. No planning is required in the application phase, only the graph matching and the classification algorithm have to be performed. In addition, our planning framework is one step towards being fully autonomous. No human interaction is needed in any of the stages of the training nor in the planning, making KT-RASP a flexible approach which could be used for new scenarios. Another advantage of KT-RASP is its adaptability to work with different skill sets, since the knowledge transfer is the rules which are semantic representations of constraints (depending not only on the product but also on the setup).

A drawback of KT-RASP is that it provides less guaranties about the feasibility of the solution than our previous approach. With our previous method, the planner verified a solution through all the checks in the different complexity levels (which was time consuming). However our actual method can be complemented with a verification run of the solution in simulation to test its feasibility. Nevertheless, when KT-RASP is properly trained by being exposed to all the different possible constraints and problems of a domain, the feasibility of the solutions is quite precise as seen in the evaluation. With our method, there is no need for checks of inverse kinematic, or workspace analysis in the execution phase resulting in big time savings. The system only needs to do graph matching and execute an already trained classification algorithm.

The experiments have been performed in the aluminum profile domain, where all the relative angles are 90 degrees or multiples of it. In order to adapt to other domains, a change of the parameter vector is necessary. The actual representation of the assembly in a topology should be able to adapt to different domains, by decomposing the parts into surfaces and inferring relations between them. Note that the representation should be capable to characterize the different structures by the restrictions to assemble them. We plan to investigate scenarios where surfaces are no longer parallel, 3D aluminum profile structures, structures assembled with Lego blocks, and actual modular products in the future.

VIII. Conclusions

In this paper we introduced a system capable to transfer knowledge about the different constraints an assembly may have. This avoids time consuming feasibility checks during planning phase which makes the algorithm applicable for productive assembly lines. The introduced method does not require any previous human knowledge about the particular domain being a step towards a fully autonomous system. We believe that these requirements have to be met to enable adaptive robotic assembly lines in the factory of the future.

References

[1] F. Suárez-Ruiz, X. Zhou, and Q.-C. Pham, "Can robots assemble an ikea chair?" *Science Robotics*, vol. 3, no. 17, pp. 421–426, 2018.

[2] I. Rodriguez, K. Nottensteiner, D. Leidner, M. Kaßecker, F. Stulp, and A. Albu-Schaffer, "Iteratively refined feasibility checks in robotic assembly sequence planning," *IEEE Robotics and Automation Letters*, 2019.

[3] M. F. F. Rashid, W. Hutabarat, and A. Tiwari, "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches," *The Int. Journal of Advanced Manufacturing Technology*, vol. 59, no. 1-4, pp. 335–349, 2012.

[4] G. Dini and M. Santochi, "Automated sequencing and subassembly detection in assembly planning," *CIRP Annals-Manufacturing Technology*, vol. 41, no. 1, pp. 1–4, 1992.

[5] U. Thomas, T. Stouraitis, and M. A. Roa, "Flexible assembly through integrated assembly sequence planning and grasp planning," in *IEEE Int. Conference on Automation Science and Engineering (CASE)*. IEEE, 2015, pp. 586–592.

[6] K. Nottensteiner, T. Bodenmueller, M. Kassecker, M. A. Roa, A. Stemmer, T. Stouraitis, D. Seidel, and U. Thomas, "A complete automated chain for flexible assembly using recognition, planning and sensor-based execution," in *Proceedings of 47st Int. Symposium on Robotics (ISR)*, 2016, pp. 1–8.

[7] R. Andre and U. Thomas, "Anytime assembly sequence planning," in *Proceedings of 47st Int. Symposium on Robotics (ISR)*, 2016, pp. 1–8.

[8] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Miguelanez, "An IEEE standard ontology for robotics and automation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1337–1342.

[9] M. Tenorth and M. Beetz, "Representations for robot knowledge in the knowrob framework," *Artificial Intelligence*, vol. 247, pp. 151–169, 2017.

[10] S. R. Fiorini, J. L. Carbonera, P. Gonçalves, V. A. Jorge, V. F. Rey, T. Haidegger, M. Abel, S. A. Redfield, S. Balakirsky, V. Ragavan, *et al.*, "Extensions to the core ontology for robotics and automation," *Robotics and Computer-Integrated Manufacturing*, vol. 33, pp. 3–11, 2015.

[11] D. Beßler, M. Pomarlan, and M. Beetz, "Owl-enabled assembly planning for robotic agents," in *Proceedings of the 2018 International Conference on Autonomous Agents AAMAS*, Sweden, 2018.

[12] A. Perzylo, N. Somani, M. Rickert, and A. Knoll, "An ontology for cad data and geometric constraints as a link between product models and semantic robot task descriptions," in *2015 International Conference on Intelligent Robots and Systems*. IEEE, 2015, pp. 4197–4203.

[13] A. Perzylo, N. Somani, S. Profanter, A. Gaschler, S. Griffiths, M. Rickert, and A. Knoll, "Ubiquitous semantics: Representing and exploiting knowledge, geometry, and language for cognitive robot systems," in *IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2015.

[14] Y. Liu, S. Mahdi Shamsi, L. Fang, C. Chen, and N. Napp, "Deep q-learning for dry stacking irregular objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), October 1-5, 2018, Madrid, Spain*, 2018.

[15] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 24-28, 2017, Vancouver, BC, Canada*, 2017.

[16] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," 2019.

[17] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, vol. 5, 16–18 Apr 2009, pp. 488–495.

[18] T. Horváth, T. Gärtner, and S. Wrobel, "Cycc pattern kernels for predictive graph mining," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04, 2004, pp. 158–167.

[19] B. Gaüzère, L. Brun, D. Villemin, and M. Brun, "Graph kernels based on relevant patterns and cycle information for chemoinformatics," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 2012, pp. 1775–1778.

[20] K. M. Borgwardt and H. P. Kriegel, "Shortest-path kernels on graphs," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, Nov 2005, pp. 8 pp.–.

[21] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[22] G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis, "Kernel graph convolutional neural networks," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 22–32.

[23] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.