Industrial Robot: An International Journal, vol. 43 no. 5, pp. 524-534

Learning of assembly constraints by demonstration and active exploration

Aljaž Kramberger
Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia
Rok Piltaver
Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana, Slovenia
Bojan Nemec
Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia
Matjaž Gams
Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana, Slovenia
Aleš Ude
Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia

Abstract
Purpose – In this paper we propose a method for learning robotic assembly sequences, where precedence constraints and object relative size and location constraints can be learned by demonstration and autonomous robot exploration.
Design/methodology/approach – In order to successfully plan the operations involved in assembly tasks, the planner needs to know the constraints of the desired task. In this paper we propose a methodology for learning such constraints by demonstration and autonomous exploration. We investigated the learning of precedence constraints and object relative size and location constraints, which are needed to construct a planner for automated assembly. In the developed system, the learning of symbolic constraints is integrated with low–level control algorithms, which is essential to enable active robot learning.
Findings – We demonstrated that the proposed reasoning algorithms can be used to learn previously unknown assembly constraints that are needed to implement a planner for automated assembly. Cranfield benchmark, which is a standardized benchmark for testing algorithms for robot assembly, was used to evaluate the proposed approaches. We evaluated the learning performance both in simulation and on a real robot.
Practical implications (if applicable) – Our approach reduces the amount of programming that is needed to set up new assembly cells and consequently the overall set up time when new products are introduced into the workcell.
Originality/value – In this paper we propose a new approach for learning assembly constraints based on programming by demonstration and active robot exploration to reduce the computational complexity of the underlying search problems. We developed algorithms for success/failure detection of assembly operations based on the comparison of expected signals (forces and torques, positions and orientations of the assembly parts) with the actual signals sensed by a robot. In this manner all precedence and object size and location constraints can be learned, thereby providing the necessary input for the optimal planning of the entire assembly process.

Keywords Assembly sequence planning, Adaptation of robot trajectories, Error detection, Learning of assembly constraints

Paper type Research paper

1. Introduction

Programming of industrial robot assembly tasks involves specifying low-level control algorithms to perform assembly operations (Nevins & Whitney, 1980) as well as the specification of an appropriate sequence of assembly operations that lead to the final product assembly (Jimenez, 2013). Currently, the initial set-up of a new automated assembly cell is a tedious process, but programming by demonstration (Argall et al., 2008) and active robot learning have been suggested recently as effective means to speed up the programming of both low-level control algorithms and high-level assembly sequences (Krüger et al., 2014). These recent research results are an example of increased introduction of artificial intelligence methods into robotics (Bogue, 2014).

Typically, programming of low-level control and high-level planning algorithms have been considered as separate problems. In our own work we showed how programming by demonstration (PbD) and active learning can be used to efficiently teach peg-in-hole and other assembly operations (Abu-Dakka et al., 2014; 2015). Other researchers showed the effectiveness of programming by demonstration and active learning for teaching assembly plans in a virtual (Martinez et al., 2014) and real environment (Kuniyoshi et al., 1994; Ahmadzadeh et al., 2015). Planning knowledge is often also provided by spoken dialogue (Breazeal et al., 2006; Lauria et al., 2002) as well as interaction in the human–robot domain (Nicolescu et al. 2001, Agostini et al. 2011) for providing expert knowledge to the robot system when it is required. In this paper we build on the results from (Abu-Dakka et al., 2014; 2015) and propose new algorithms for learning assembly sequences and constraints with a real robot. The main novelty of our approach compared to other methods proposed in the PbD literature lies in the application of low-level sensorimotor knowledge (assembly operations acquired by PbD) to the learning of higher-level concepts (assembly sequences and constraints) by active exploration.

According to (Jimenez, 2013), an assembly is an object composed of individual parts in given relative placements, such that they do not overlap and each part is touching a subset of the assembly. The aim of assembly sequencing, which is part of a broader problem of assembly planning, is to compute an ordering of assembly operations that bring the individual parts together, given a description of their final positions in the assembled product. Assembly sequencing is usually treated as a combinatorial problem, which deals with symbols corresponding to subassemblies or parts. It includes three steps: the definition of precedence constraints, the generation of all feasible sequences and the selection of the optimal sequence. Our approach aims at reducing setup-times of robot cells for automated assembly. Long setup-times represent the main obstacle to wider application of robotized solutions for assembly problems.

The determination of feasible assembly sequences is the result of applying a search and/or optimization algorithm in the space of possible assemblies. Exhaustive search is the simplest strategy ensuring completeness, but is impractical due to a high computational cost, except for very simple assemblies. Heuristic graph search strategies (Zhao & Masood, 1999), simulated annealing, neural networks, genetic and others algorithms are therefore used instead (Russel & Nordvig, 2010). Other approaches apply swarm optimization methods to solve this problem (Wang et al., 2010). Assembly planning can be successfully accomplished if all necessary features of all assembly entities are known. (Eng et al., 1999) proposed an approach where features are extracted from CAD models. More advanced approaches like answer set programming (ASP) (Gelfond 2012) have been developed in AI to deal with difficult, typically NP-hard problems. In our work we instead use programming by demonstration to identify feasible sequences, thereby reducing the complexity of the search problem to rather simple operations on precedence graphs.

In the following we analyse the following two key problems that need to be solved to generate correct assembly sequences: 1) learning of precedence constraints from a symbolic representation of human demonstrations of feasible assembly sequences and 2) learning by active exploration of relative sizes and locations of parts and holes involved in the assembly. The proposed algorithms are described in Section 2. The robot control and detection algorithms that are needed to realize active exploration for learning of relative size and location constraints are explained in Section 3. The experimental results are discussed in Section 4, followed by the conclusion in the final section.

2. Planning actions and learning of the assembly constraints

In order to successfully plan the operations involved in an assembly task, the planner needs to know the constraints of the given task. We assume that these constraints are not known in advance, hence they have to be learned. In this section, we describe a procedure to learn the precedence and relative size constrains of a given assembly task. The result can then be used to generate a set of assembly sequences (Ramos et al., 2011) and explore it in order to find feasible and possibly optimal sequences. The first set of constraints to be learned are the precedence constraints between the assembly operations where each assembly operation corresponds to placing a part in its final position in the assembly. The learning algorithm uses a set of feasible assembly sequences provided by demonstration. It generalizes from the maximally restrictive constraints corresponding to each of the given feasible assembly sequences to a less restrictive set of constraints using logical operations. The algorithm is described in Section 2.1 and guaranties that all assembly plans generated from the generalized constraints produce feasible assembly sequences.

The second set of constrains to be learned define the relation between the assembly parts and their final locations. This information is sometimes given explicitly by the relative placements of individual parts in the goal assembly or can be detected by a vision system. However, vision system errors may prevent identifying individual parts and precise coordinates of their final positions. In addition, a given part may be placed at several locations, which results in a multitude of feasible assembly sequences that also consider location of the parts and not only the order of placing them in the predefined locations. Therefore, we propose to learn the constraints defining the feasible positions for each part using experiments performed by the robot. The algorithm that plans the experiments and learns the constraints based on the results of the previous experiments is presented in Section 2.2.

2.1. Generalizing precedence constraints from assembly sequences

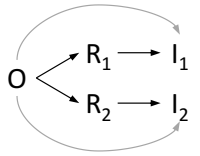
The algorithm for generalizing assembly sequences from a given set of feasible sequences can be illustrated with a task of replacing two batteries in a remote control (

Figure 1). Actions that need to be executed in order to complete the battery replacement task are: open the battery door (O), remove battery 1 and 2 (R_1, R_2), and insert new batteries 1 and 2 (I_1, I_2).

Figure 1: The task of replacing batteries in a remote control requires the following actions: open the battery door, remove each of the two batteries and replace them with new ones.

Note that exactly one action is required for each of the manipulated parts in the remote control assembly. The precedence relation between the actions is defined by the following two constraints: the battery door has to be opened first, a new battery can be inserted only after the old battery that initially occupies the final position of the new battery has been removed. The relation describing the precedence constraints is termed the *must-precede relation.* It can be represented with a set of ordered pairs, a matrix (used in the algorithm), or a graph (used as human readable output) shown in Fig. 2. Must-precede relation is transitive, therefore its transitive reduction can be used instead of the original relation to simplify the representation and make it more readable. For example, an edge from action O to action I_1 in the graph of the original relation means that action O must be executed before action I_1 can be executed. This constraint is represented by the path from action O to action I_1 in the transitive reduction of the relation – shown with black arrows in Fig. 2. All ones in the matrix, arrows in the graph, and pairs in the list that are not in the transitive reduction of the relation are shown in grey.

	O	R_1	R_2	I_1	I_2
O	0	**1**	**1**	1	1
R_1	0	0	0	**1**	0
R_2	0	0	0	0	**1**
I_1	0	0	0	0	0
I_2	0	0	0	0	0



{(O, R_1), (O, R_2), (O, I_1), (O, I_2), (R_1, I_1), (R_1 , I_2)}

Figure 2: The must-precede relation that defines the precedence relation between actions in battery replacement task represented with a matrix, graph, and list of ordered pairs.

The algorithm for generalizing assembly sequences (Algorithm 1) executes one iteration for each given feasible assembly sequence s with n actions. Each iteration first computes the maximal set of precedence constraints C_s between the actions of the considered assembly sequence s (lines 3-8): if an action s_i precedes an action s_j in the assembly sequence s, then the pair is added to the relation C_s. Examples for two plans are shown in Fig 3. After that, C_s is merged with the currently known approximation of the *must-precede* relation C in order to generalize it (line 9). This is achieved by computing the intersection of the two relations according to the following equation:

$$C'(i,j) \Leftrightarrow C(i,j) \wedge C_s(i,j) \quad (1)$$

The result of merging the maximal constraints for the two assembly sequences shown in Fig. 3 is shown in Fig. 4. The graph of the merged relation includes exactly the edges that are present in both graphs of the merged relations. For example, action O precedes action R_1 in both assembly sequences shown in Fig. 3, therefore the merged relation also includes this constraint. On the other hand, action R_1 precedes action R_2 only in one of the two assembly sequences (Fig. 3b), therefore it is not included in the merged relation. By observing a feasible assembly sequence in which an action precedes another action, the algorithm learns that the first action can precede the second and therefore removes the related constraint from the set of constraints thus the new set of constraints is more general, i.e. enables planning of more assembly sequences.

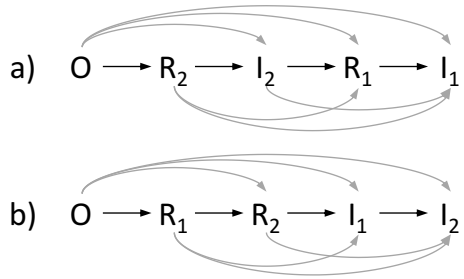


Figure 3: Examples of two maximal set of constraints that make the assembly sequences O, R_2, I_2, R_1, I_1 (a) and O, R_1, R_2, I_1, I_2 (b) feasible. The edges that are not in the transitive reduction of the relations are shown in grey.

Algorithm 1: generalize assembly sequences

Input: set of feasible assembly sequences *S*, each with *n* actions

```
C = ones(n, n);
for each s in S
    C_s = zeros(n, n);
    for i = 1 to (n - 1)
        for j = (i + 1) to n
            C_s(s_i, s_j) = 1;
        end for;
    end for;
    C = C && C_s;
end for;
return C;
```

Output: *C* – a matrix representing the *must-precede* relation (precedence constraints)

If no feasible sequences are given to the algorithm, it will return a matrix filled with ones as the must-precede relation; no feasible plans can be planed based on it. If one assembly sequence is given to the algorithm, it will return the maximal set of constraints that make the given plan feasible; exactly the given assembly sequence can be planed based on it. The algorithm will generalize the set of constraints further for each additional given assembly sequence until the correct set of constraints is learned. The set of feasible assembly sequences that can be planed based on the learned must-precede relation includes all the given assembly sequences and possibly additional assembly sequences. For example, the learned set of constraints shown in Fig. 4 enables planning assembly sequences: O, R_1, R_2, I_2 I_1; O, R_2, R_1, I_1 I_2; and O, R_2, R_1, I_2 I_1 in addition to the two assembly sequences (shown in Fig. 3) from which the set of constraints was generalized. In this example, two assembly sequences were generalized to the set of constraints that enables planning five feasible assembly sequences.

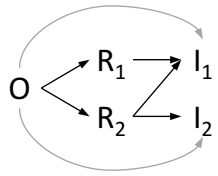


Figure 4: The must-precede relation computed as the intersection of maximal precedence constraints for the two assembly sequences shown in Fig. 3. The edges that are not in the transitive reduction of the relations are shown in grey.

2.2. Learning relative size of parts and holes using autonomous robot exploration

The second part of the algorithm deals with geometrical constraints between the parts and their final locations. It learns which peg fits into which hole starting from no knowledge except for the list of pegs and holes for which the relation is to be learned. The proposed algorithm learns from experiments executed by a real robot, which tries to insert the available pegs into different holes. If the action is executed successfully, the algorithm learns that the peg fits in the hole; otherwise it assumes that the peg is too big to fit into the hole. For this learning algorithm to work, the robot should be able to recognize unsuccessful actions. In our system, this is

accomplished using force/torque data and poses extracted by vision (see Section 3). The proposed algorithm obtains new knowledge from each experiment and plans the next experiment until the complete relation is learned. The goal is to learn the complete relative size relation with as few experiments as possible.

part/hole	R1	R2	S1	S2	BR
P1	1	1	1	1	1
P2	1	1	1	1	1
P3	0	0	1	1	1
P4	0	0	1	1	1
SH	0	0	0	0	1

Figure 5: The smaller-than relation between parts (P1, P2, P3, P4 - pegs, SH – shaft) and holes (R1, R2 - round holes, S1, S2 – square holes, BR – big round hole in the back plate) of the Cranfield assembly (see Fig. 8).

The knowledge of the learning algorithm is represented by the transitive *smaller-than* relation between the set of holes and the set of parts, which describes the constraints regarding the possible final locations of the parts in the assembly. Fig. 5 shows the matrix representing the smaller-than relation for the Cranfield assembly (see Fig. 8), e.g. part P1 fits in all the holes while part SH fits only in the BR hole. The *bigger-than* relation, which is the inverse of smaller-than relation, could be used as well. Note that the robot can only detect if a part is smaller than a hole or not; it cannot detect if they are of exactly the same size. Therefore, *is-equally-big*, is *greater-or-equal*, and is *smaller-or-equal* are not appropriate representations of the knowledge.

The smaller-than relation is extended to a relation on the union of the sets of parts and holes as shown in Fig 6. This is done by first setting the relation between parts and holes as in the original relation and then calculating the transitive closure of the relation. In this way the relative sizes between individual parts and holes are obtained. For example, Fig. 6 shows that parts P1 and P2 are smaller than parts P3, P4 and SH; or that hole S1 is bigger than hole R1 and smaller than hole BR.

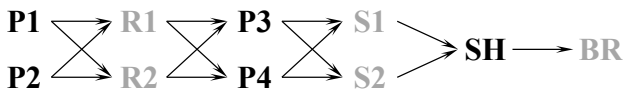


Figure 6: The graph of transitive reduction of the smaller-than relation on the union of the sets of parts (black) and holes (grey) of the Cranfield assembly: each element is smaller than all the elements that can be reached from it following a directed path in the graph.

The robot cannot compare the sizes of two parts or two holes directly; therefore, the algorithm needs to represent its knowledge with the extended relation, which enables determining the relative sizes based on transitivity. In general, the rules given by Eqs. (2) and (3) that determine relative sizes of parts (and holes) apply for any given part P_i, P_j and hole H_k, H_l. Eq. (2) can be interpreted as: if part P_i fits into a hole but part P_j does not, then the part P_i is smaller than part P_j. Eq. (3) can be interpreted as: if a part fits into hole H_k but does not fit into hole H_l, then the hole H_l is smaller than hole H_k.,

$$P_i < H_k \wedge H_k < P_j \Rightarrow P_i < P_j \tag{2}$$

$$H_l < P_i \wedge P_i < H_k \Rightarrow H_l < H_k \tag{3}$$

After the relative sizes of the parts and holes are determined, the transitivity property is used to infer whether a part is smaller than a hole, without the robot actually performing the corresponding experiment. For example, knowing that part P1 is smaller than part P3 and that part P3 fits in the hole S1 implies that part P1 also fits in the hole S1 (Fig. 6). In general, the rules given by the Eq. (4) and (5) apply for any given part P_i, P_j, P_k and hole H_l. Eq. (4) can be interpreted as: if a bigger part fits into a hole, then the smaller part fits into the same hole, too. Eq. (5) can be interpreted as: if a smaller part does not fit into a hole then the bigger part does not fit into the same hole neither.

$$(P_i < P_j) \wedge (P_j < H_l) \Rightarrow P_i < H_l \tag{4}$$

$$(H_l < P_j) \wedge (P_j < P_k) \Rightarrow H_l < P_k \tag{5}$$

When no more relations can be determined theoretically, the next experiment to be performed by the robot is suggested using a heuristic function, which is based on Eq. (4) and (5) as explained below. Each executed experiment provides information about the relative size of part P_j and hole H_l, therefore only

experiments with part-hole pairs for which the relative size is not yet known are performed. In addition, the size of part P_i or P_k relative to the hole H_l is determined by the same experiment if the result of the experiment matches the conditions in Eqs. (4) or (5), respectively. Experiments with part-part-hole triplets that can match either of the two equations and for which only the relative sizes of parts are known are selected. Performing an experiment with a part-hole pair (P_j, H_l) that fits the conditions of both Eqs. (4) and (5) is guaranteed to determine at least one additional relative size of another part (P_i or P_k) and the hole H_l. Therefore experiments with part-hole pair (P_j, H_l) that match the template shown in Figure 7 and described by Eqs. (4) and (5) are preferred:

	P_i	P_j	P_k	H_l
P_i	0	1	1	?
P_j	0	0	1	**?**
P_k	0	0	0	?
H_l	?	**?**	?	0

Figure 7: Performing experiment with part-hole pair (P_j, H_l) matching the above template will determine the unknown relative size of the two (bold question mark) and an additional relative size between part P_i or P_k and the hole H_l (grey question marks) based on transitivity of the smaller-than relation.

The heuristic function counts how many different assignment of parts P_i and P_k match the template for each possible (P_j, H_l) part-hole pair. The experiment with part-hole pair with the highest count is executed. In case of a draw, the one with the highest count based on Eq, (2) and (3) is used. In case of another draw, the pair for which the relative size of the part is known for the highest number of holes. In case of another draw, the pair for which the relative size of the hole is known for the highest number of parts. Finally the part-hole pairs with lower part and hole indices are selected.

Note that the algorithm assumes that the shapes of pegs and holes are such that if a peg fits into a hole then it fits into all larger holes, and it does not fit into any smaller hole, all smaller parts fit into this and all bigger holes and none of the bigger parts fit into this or any smaller hole. This is for instance true for any set of round pegs and holes as well as for the five parts that fit into the holes in the back plate of the Cranfield assembly (Collins et. al 1985) shown in Fig. 8. After the smaller-than relation is learned, it is used to compute which parts can be put in which hole in order to assemble the parts. The holes are assigned to the parts using a greedy iterative approach: the biggest unassigned part is assigned to a non-assigned hole that is big enough for the part until all parts have an assigned hole. If there is no hole that is big enough for a given part at any step of the algorithm, the assembly is recognized as non-solvable. Otherwise, the algorithm can determine any feasible set of final locations of the parts.

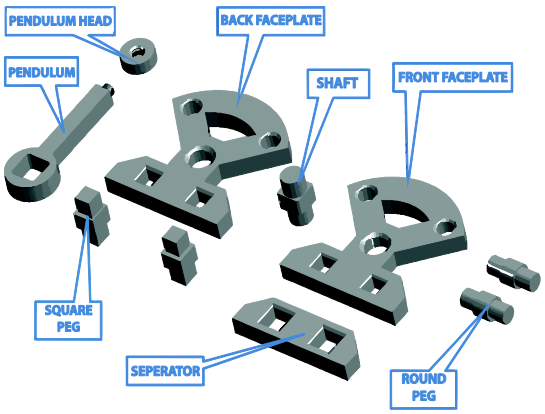


Figure 8: Parts of the Cranfield assembly

3. Revising action and assembly constraints by robot exploration

This section presents a framework to verify that a sequence of assembly operations generated by the planner is executable on a real robot setup. This can be achieved by attempting to carry out the generated assembly plan and monitoring the success of execution.

To perform the desired assembly plan, we have to teach the robot how to execute all relevant assembly operations. Since in our real experiments we investigated the teaching of assembly plans consisting of different peg-in-hole like actions, we limit our discussion here to such operations. Robust peg-in-hole execution is best accomplished using force control (Nevins & Whitney, 1980), hence both positions and orientations as well as the resulting forces and torques need to be considered when teaching the required operations (Rozo et al., 2013). In Section 3.1 we explain how such assembly operations can be taught by human demonstration and adopted with respect to the required forces and torques. Task description and failure detection are explained in Section 3.2.

3.1. Teaching assembly operations

This section gives a brief overview of the basic procedure for learning task specific trajectories by human demonstration. In this work we applied kinesthetic guiding for demonstration of the required movement trajectories, as it is one of the most intuitive methods with a quick setup time (Kormushev et al., 2011). The operator simply grabs the robot's end effector and guides it along the desired trajectory (Fig. 9). Please note that kinesthetic guiding might affect the captured forces and torques, especially when joint torque sensors are used to estimate forces and torques acting on the robot tool, therefore it is often necessary to perform two steps. In the first step, only the position and orientation trajectory is recorded, while the resulting forces and torques are obtained in the next step, where we replay the demonstrated trajectory and the resulting forces are not affected by the human demonstrator. The recorded force-torque trajectories represent a reference for later adaptation. After the recording of the position and orientation trajectories, they are encoded as Cartesian Space Dynamic Motion Primitives (DMPs) (Ijspeert et al., 2013; Ude et al., 2014). Since forces and torques are used only as desired variables along the trajectory and not as robot control variables, they do not need to be encoded by DMPs. Instead we use a linear combination of radial basis functions (Abu-Dakka et al., 2015) to represent them.

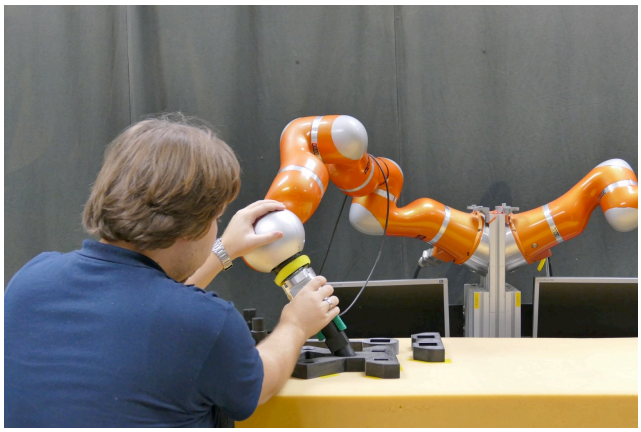


Figure 9: Teaching the example movement library with kinesthetic guiding.

When executing the demonstrated actions, the actual forces and torques can significantly differ from the forces and torques captured during the demonstration. These deviations arise mainly due to non-repeatable grasp configurations and errors in the estimated object poses. Therefore, the learned trajectories have to be adapted in order to meet the required force/torque profile during the execution. In our work we apply the method originally proposed by (Abu-Dakka et al., 2015), where the adaptation is accomplished in current execution cycle as well as in subsequent execution cycles using ILC (Iterative Learning Control) framework (Bristow et al., 2006).

3.2. Failure detection

The execution of peg-in-hole encompass the following operations:

1. Initial hole search procedure (Abu-Dakka et al., 2014).
2. Execution of the learned trajectory with on-line adaptation to the desired force/torque profile using admittance PI control law (Abu-Dakka et al., 2015).
3. Slow down of the trajectory execution on excessive force/torque deviations (Ude et al., 2014).
4. Execution of exception strategies.

To be able to successfully revise the plans and assembly constraints, it is important to detect whether the action execution was successful or not.

We implemented an execution monitor mimicking a reasoning engine (Martínez et al., 2014) to properly plan the above mentioned execution steps and evaluate the outcome signals that are generated by each method. In the first step, the exact position of the hole has to be determined. As vision-based solutions are usually insufficiently accurate, we apply a stochastic search algorithm (Abu-Dakka et al., 2014). In the second step, the actual PIH trajectory is executed, where the necessary adaptation to the desired force/torque profile is provided by admittance PI control law. On excessive deviation of the measured forces and torques, the desired position trajectory is slowed-down using DMP phase stopping mechanism (Ude et al., 2014). This gives the time to the integral part of the admittance control algorithm to adapt to the new situation.

In this work we newly implemented a detection algorithm to identify large discrepancies between the expected and actual forces and torques, which is the basis to determine the success or failure of the executed task:

$$\|\boldsymbol{F}_{mes} - \boldsymbol{F}_{des}\| \geq F_{stop} \tag{6}$$

$$\|\boldsymbol{M}_{mes} - \boldsymbol{M}_{des}\| \geq M_{stop} \quad (7)$$

Here $\boldsymbol{F}_{mes}$ and $\boldsymbol{M}_{mes}$ represent the measured forces and torques during the task execution, $\boldsymbol{F}_{des}$ and $\boldsymbol{M}_{des}$ are the desired forces and torques recorded during the human demonstration and F_{stop} and M_{stop} are the selected thresholds. If the force/torque discrepancies are too large for the adaptation algorithm to work, the execution monitor stops the execution and signals to the planer that the execution was not successful. For success detection we cannot rely on force/torque evaluation only. Another possible criterion is if the PIH trajectory has been fully executed, i.e. we compare the expected robot position after the execution of the assembly operation, here denoted by $\boldsymbol{p}_{goal}$, with the actual robot position at the end of motion, here denoted by $\boldsymbol{p}_{end}$.

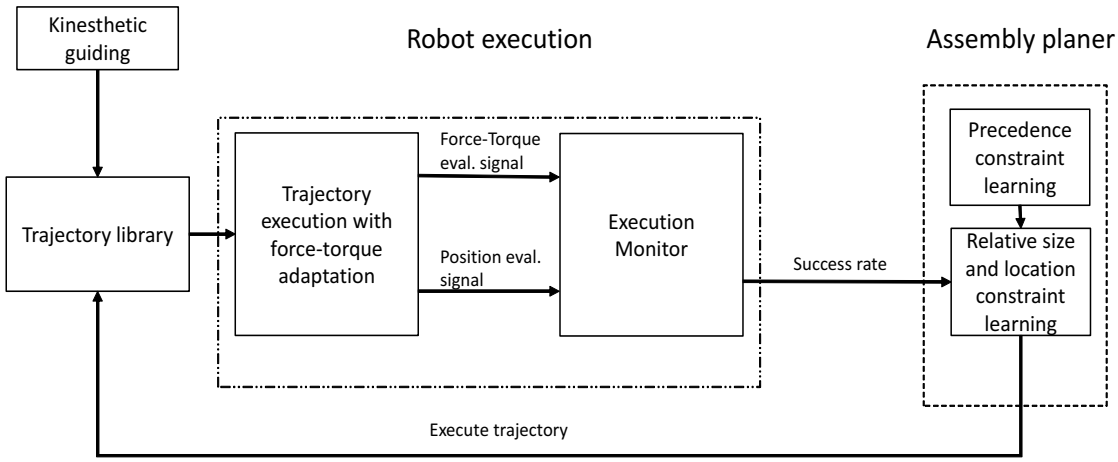


Figure 10: Overview of the complete system, which comprehends trajectory learning, trajectory library, robot execution subsystem and assembly planner.

In some cases, during the PIH execution the peg can get stuck in the hole, which results in increased forces violating conditions (6) and (7), although the peg fits the hole (Fei and Zhao, 2005). In such cases the execution is interrupted and the robot applies an exception strategy. Our exception strategy is inspired by how humans solve the peg jamming problem. When observing human performance, we noticed that humans normally start rotating and shaking the object, which in most cases releases the peg and enables the continuation of the insertion task. Therefore, we applied a sinusoidal force pattern with amplitude of 10N and frequency of 2Hz to the robot gripper around each of three principal axes in the tool coordinate system. After that, we tried to continue the previously interrupted operation while observing the resulting forces and torques. In the majority of cases, this exception strategy successfully resolved the peg jamming problem.

The proposed failure detection procedure may in some cases produce false negatives, i.e. if the robot fails to insert the peg, but the peg actually fits into the hole. In order to overcome this problem, we repeat the execution procedure steps 1-4 (explained above) if the insertion was not successful. Experimental results provided in the next section (see Fig. 18) demonstrate, that we obtain 100% success in failure detection already with two repetitions. If the insertion was not successful after two repetitions, we conclude that the object does not fit into the hole. Fig. 10 gives an overview of the proposed learning system.

4. Experimental evaluation

In order to evaluate the performance of the developed algorithms, we used a classical assembly called Cranfield benchmark (Collins et al., 1985) shown in Fig. 8. The Cranfield benchmark distinguishes itself by its compactness and portability, ability to test a variety of assembly operations and can be assembled by a majority of the current robotic systems. In the following we discuss the performance of the proposed algorithms when applied to the Cranfield assembly.

4.1. Results on learning precedence constraints

The learning of precedence constraints was evaluated on the Cranfield assembly, in which the parts shown in Fig. 8 have to be assembled into a single object. The precedence constraints (Fig. 11) of the Cranfield assembly can be defined analytically and allow 5320 feasible assembly sequences. The back plate (BP) must be

put in place before pegs (P1, P2, P3, P4) or shaft (SH) can be put in place. Square pegs P3 and P4 must be put in place before separator SP and shaft SH must be placed before pendulum PD. Front plate FP is put in place last. Pendulum head PH must be screwed onto the pendulum before pendulum is put in its place.

The evaluation method is as follows: a set of m feasible, randomly generated assembly plans are given as input, the proposed algorithm processes them and outputs the must-precede relation, then the number of assembly sequences that can be generated based on the learned must-precede relation is computed. The random generator of feasible sequences uses the precedence graph shown in Fig. 11. The procedure is repeated 1000 times for each number m and the result is used as a sample to estimate the probability that the algorithm generalizes the m given sequences to M assembly sequences, where M is discretized with step 200. The probability is estimated because computing the exact probability that the algorithm generalizes from m to M is infeasible. For example, the algorithm would have to be used $(5320 \cdot 5319 \cdot 5318)/(2 \cdot 3) > 25 \cdot 10^9$ times for $m = 3$, while the number increases exponentially for larger m. In addition, our experiments showed that the estimated probability does not fluctuate much after a couple of hundred random sets of input assembly sequences are analyzed, which indicates that it converged near the true value of the probability.

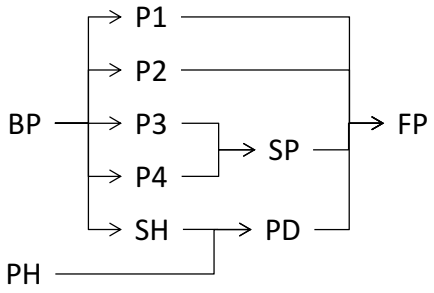


Figure 11: Transitive reduction of the must-precede relation for the Cranfield assembly: if there is a path from part A to part B, part A must be put in place before part B can be put in its place.

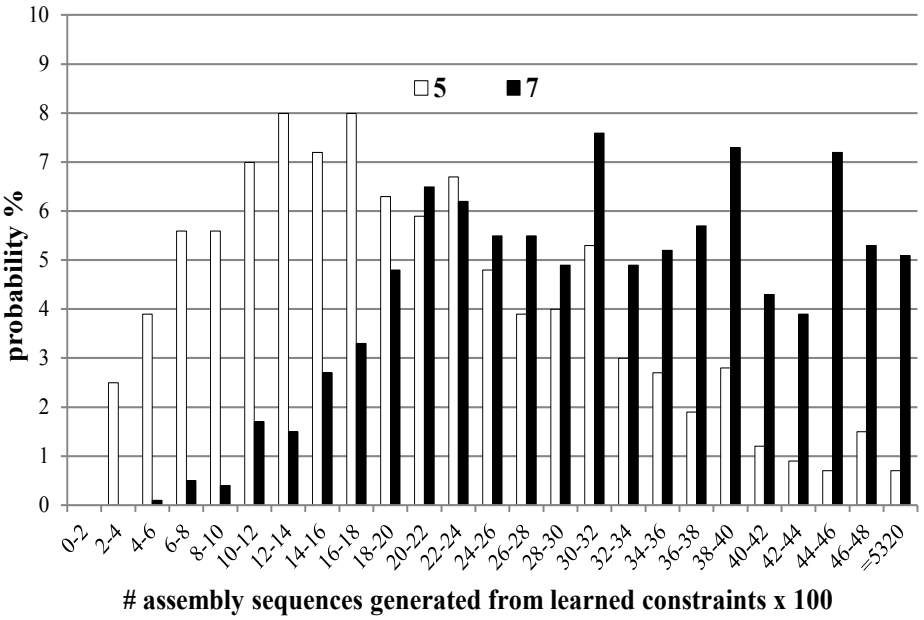


Figure 12: Probability of generalizing to a given number of assembly sequences from 5 or 7 random feasible Cranfield assembly sequences

The probability of generalizing 7 random feasible assembly sequences to more than half of all possible assembly sequences is higher than 50 %. If ten random assembly sequences are given as an input, the algorithm generalizes to ~75 % of all feasible assembly sequences on average. If 20 or 30 assembly sequences are used as the input, the algorithm generalizes to all possible assembly sequences with 72 % or 94 % probability, respectively.

More detailed results are shown in Fig. 12 and 13. In both figures, $M/100$ is given on the horizontal axis with the step of 200 – this represents the number of assembly sequences that can be constructed based on the learned must-precede relation. The last column corresponds to generalizing to all the feasible assembly sequences. The vertical axis corresponds to the probability of generalizing to the given number of assembly sequences. Shading of the bars corresponds to the number of assembly sequences m used as input to the algorithm: 5 or 7 in Fig. 12 and inputs 15, 20 and 30 in Fig. 13.

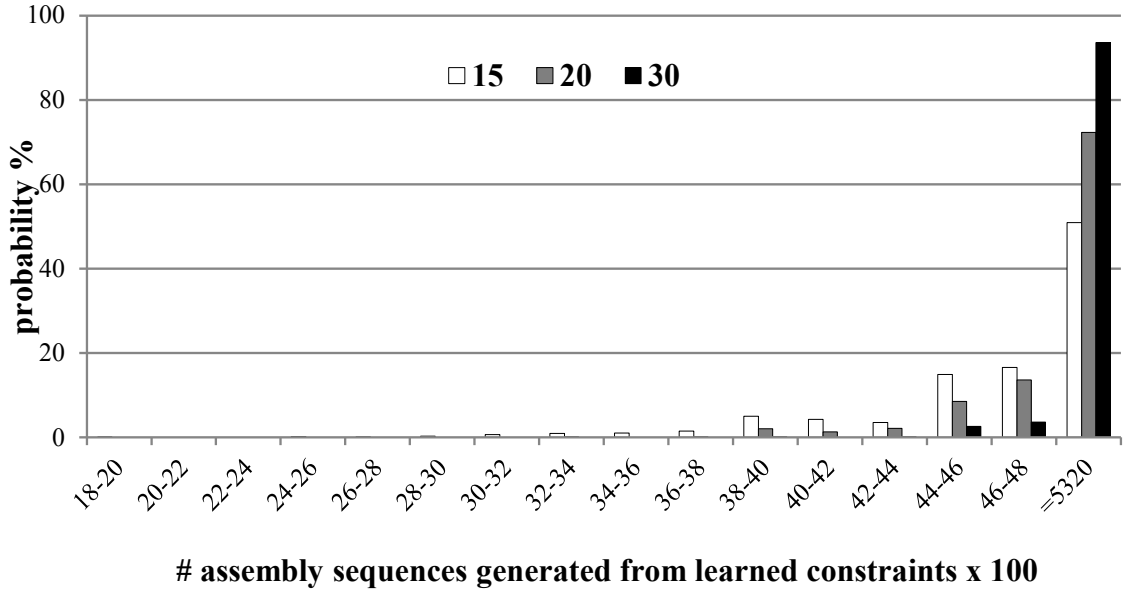


Figure 13: Probability of generalizing to a given number of assembly sequences from 15, 20 or 30 random feasible Cranfield assembly sequences.

The above results are valid for random sets of feasible plans, which are likely to contain redundant information. Note that non-redundant sets of plans would enable faster learning: generalizing to more plans or from fewer input plans. As part of our future work, we intend to evaluate and improve an interactive learning algorithm that guides the user towards providing the most informative learning plans.

Note that all the sequences that can be generated from the learned must-precede relation are feasible. The following is a sketch of the proof. Since the input assembly sequences are feasible they respect all the precedence constraints C and the maximal set of constraints C_s generated from each given assembly sequence s are therefore a superset of the true precedence constraints C. It follows that intersection of such a maximal set of constraints C_s over all given feasible assembly sequences is also a superset of the true set of precedence constraints C, therefore all plans that respect the constraints given by the intersection also respect all the true constraints and are therefore feasible.

$$\forall s: C \subseteq C_s \Leftrightarrow C \subseteq \cap_s C_s \tag{10}$$

The generalization algorithm has quadratic time complexity in the number of parts (actions) and linear time complexity in the number of given learning plans. This makes it applicable to real-life problems that may be considerably more complex than the Cranfield benchmark.

4.2. Learning the relative object size and location using a real robot

Evaluation of the size- and object location learning algorithm was performed with five parts (two square pegs and three round pegs at two different sizes) that need to be put into five holes of the back plate (Cranfield benchmark, see Fig. 8).

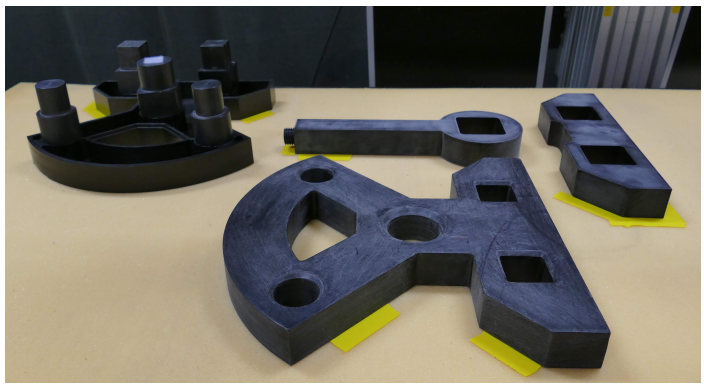


Figure 14: Setup of the work place: the front plate (FP) in the back of the figure represents the object container and the back plate (BP) in the front of the figure represents the assembly base.

Kuka LWR-4 robot arm with gravity compensation, controlled through the FRI interface (Schreiber et al., 2010), was used in these experiments (see Fig.

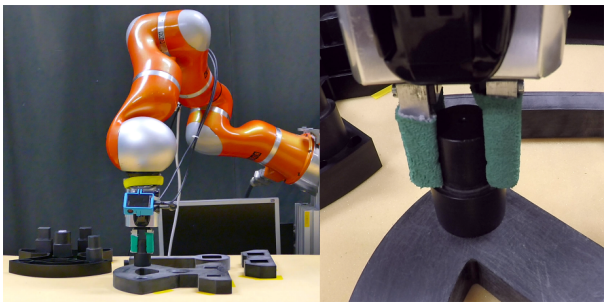


Figure 15: Successful insertion of the round peg in the small round hole.

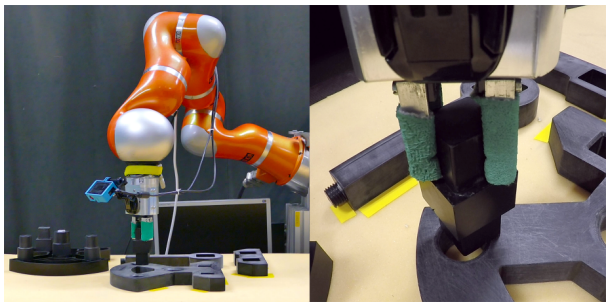


Figure 16: Unsuccessful insertion of square peg into the small round hole.

9). Kuka LWR-4 is a 7 DOF robot, equipped with a two-finger gripper RH-707 capable of grasping every objects in the Cranfield assembly. Kuka LWR-4 has a torque sensor mounted in every joint, therefore no external force-torque sensor were needed in our experiments. Torque data measured in joints are transformed to Cartesian forces and torques in real-time using the dynamic model of the robot. The test environment to perform the robot-based size and location learning was set as shown in Fig. 14. All of the acquired objects were placed in the work area of the robot so that they can be easily accessible by the robot arm when needed. The smaller objects, e.g. square pegs, round pegs and shaft, were placed on the front plate that acts as a part container. In this experiment, the locations of containers were determined by vision (Buch et al. 2013). After every execution, the objects had to be placed back to the correct starting place. Manipulation tasks were performed in joint space with high stiffness to achieve appropriate position accuracy. Even this set-up could not prevent uncertainties due to grasping and position discrepancies that are the outcome of vision errors and objects moving in the hand while being manipulated. For this reason, the proposed search and force-torque strategies adaptation were applied to insert the pegs, followed by success/failure detection.

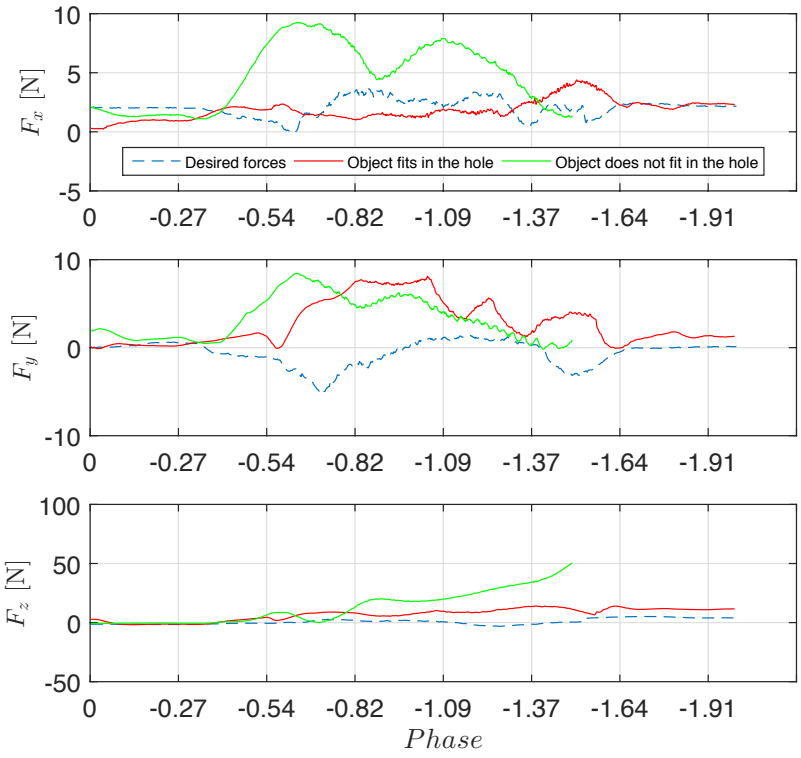


Figure 17: Forces arising during the successful round peg insertion (red) and unsuccessful square peg insertion (green) compared to the desired forces (blue), all plotted with respect to the DMP phase. Note that in case of square peg insertion, the execution monitor interrupted the action before the trajectory was completed, as it determined that the object cannot fit the hole.

The forces arising during the insertion step can be seen in Fig. 17. The blue dotted line corresponds to the desired force profile of the learned insertion. The red solid line represents the insertion force during successful execution (Fig. 15). If the object was too large to fit into the specified hole, the measured forces and torques would violate the conditions set in (6) and (7). Moreover, the robot end position does not coincide with the goal position (not shown in the graph), signalling to the learning system that the execution has failed. An example execution failure is shown in Fig. 16. The proposed concept was evaluated on five parts of the Cranfield benchmark.

In addition, several insertion operations with square, round, tight round peg, and shaft were performed to evaluate the robustness of the execution method. Only PiH operations with the peg actually fitting into the hole were carried out in this experiment. In total 360 PIH operations were performed, with randomly perturbed initial position. Therefore, the robot had to start by searching for the hole, then perform the insertion operation, and finally evaluate if the object was successfully placed into the whole. Fig. 18 shows the success ratio of insertion experiments with different objects. Furthermore, we also evaluated the method on inserting a tight peg (results presented in Fig. 18) where the tolerances between the hole and peg were 0.1 mm, compared to the regular peg, where the tolerance was 1.3 mm. The obtained results from the tight peg insertion are comparable with results from the normal peg insertion, consequently showing the robustness of the proposed method.

Object	Hole	No. of executions	No. of successful executions in the 1. trial	No. of successful executions in the 2. trial	Success ratio after second execution (%)
Round peg	H1	40	40	40	100
	H3	40	35	40	100
	H4	40	39	40	100
Tight round peg	H1	40	40	40	100
	H3	40	36	40	100
	H4	40	39	40	100
Square peg	H3	40	39	40	100
	H4	40	39	40	100
Shaft	H3	40	40	40	100

Figure 18: Success ratio of the proposed evaluation algorithm for determening relative size of objects and holes, where H1 is the small round, H3 big round and H4 square hole of the back plate.

A naive approach requires 25 experiments (try to put each of the five parts in each of the five holes) to learn the smaller-than relation. Systematic order of experiments (experiments with parts and holes with lower indices first) combined with reasoning based on the transitivity requires at least 17 experiments (with probability 6.66%), 20.53 experiments on average, and at most all of the 25 experiments (with probability 4.11%). The results were obtained by running the algorithm on each possible permutation of the parts and holes. Random order of experiments (put a random part in a random hole, choosing only from part-hole pairs with currently unknown relative sizes) combined with reasoning based on transitivity requires at least 15 experiments (with probability 0.09%), 20.65 experiments on average, and at most all of the 25 experiments (with probability 0.77%). Finally, the suggested algorithm requires at least 16 experiments (with probability 0.11%), 19.57 experiments on average, and at most 24 experiments (with probability 0.44). The detailed comparison of the suggested size-learning algorithm with the random order of experiments is shown in Fig. 19.

The above results are valid if the algorithm starts with no prior knowledge about relative sizes of parts and holes. Furthermore, the algorithm requires fewer actions to learn the relation if it start with some knowledge, which can be trivially extracted from a set of feasible, partial or even infeasible assembly sequences. The Video available at http://www.ijs.si/~aude/AssemblyPlanning.mp4 demonstrates one example of the developed learning process. The objects and locations were chosen by the developed reasoning algorithm. In this case the number of executions to learn the complete relation was 20. After all constrains were learned, the system can construct a planner to compute complete assembly plans.

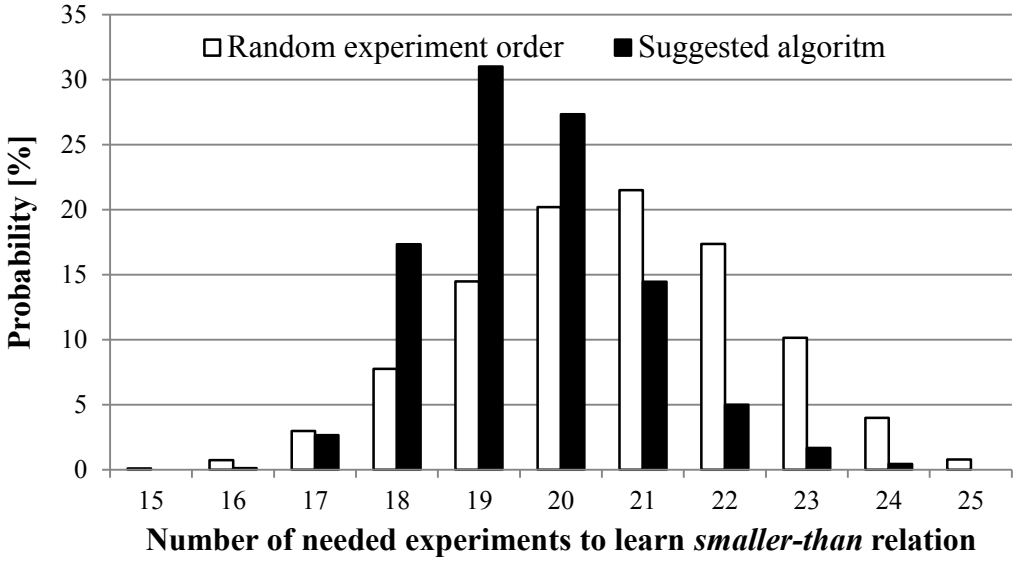


Figure 19: Comparison of the suggested size-learning algorithm with a random search.

5. Conclusion

In this paper we propose an integrated environment for learning assembly sequences and constraints. The main purpose of our work is to shorten set-up times when preparing new automated assembly tasks.

We proposed two novel algorithms that facilitate the planning of assembly tasks. The first algorithm learns the precedence constraints by generalizing a given set of assembly sequences to a set of precedence constraints, which is used to generate previously unseen assembly sequences and guarantees that all the generated assembly sequences are feasible. The initial set of feasible assembly sequences is usually acquired by human demonstration. The second algorithm learns which parts of the assembly can be put into which hole in another assembly part. Often the goal locations of the parts are given in advance, however when this information is not given, the relative part size learning can be used to learn which part fits into which hole. In this case it is necessary to integrate the real robot into the learning process. A bidirectional communication channel is established for communication between the robot system and the planner. First the planer algorithm sends the command to the robot specifying the desired part and its final location. In the second step, the robot picks the desired part from the container and tries to execute the specified action. If the hole is too small, the proposed error detection algorithm computes that the difference between the actual and the recorded forces and torques as well as the difference between the actual position of the object and the goal position is very large, indicating that the execution has failed. The execution monitor then sends the success/failure signal to the planer, which updates its knowledge base. The algorithm then chooses the next experiment with the goal to minimize the number of needed experiments in order to learn the relative sizes of parts and placements. The described learning procedure belongs to the learning-by-exploration methods since the robotic system needs to experiment on its own to become able to create the desired solutions, i.e. assemblies. The proposed algorithms have a high time complexity in terms of the number of holes and pegs, which enables the learning of constraints for assemblies of realistic proportions on real robot hardware. Secondly, they are rather general, applicable to all assemblies of similar functionality. Thirdly, they need only a small amount of domain knowledge and can therefore be applied to similar assemblies.

Acknowledgment

The research leading to these results has received funding from the EU FoF project no. 680431, ReconCell.

References

Abu-Dakka, F. J., Nemec, B., Jørgensen, J. A., Savarimuthu, T. R., Krüger, N., and Ude, A. (2015) Adaptation of manipulation skills in physical contact with the environment to reference force profiles. Autonomous Robots, 39(2):199-217.

Abu-Dakka, F. J., Nemec, B., Kramberger A., Glent Buch A., Krüger N., and Ude A. (2014) Solving peg-in-hole tasks by human demonstration and exception strategies. *Industrial Robot: An International Journal*, 41(6):575-584.

Agostini, A., Torras, C., and Wörgötter, F. (2011) Integrating Task Planning and Interactive Learning for Robots to Work in Human Environments. In International Joint Conference on Artificial Intelligence, pages 2386-2391.

Ahmadzadeh, S. R., Paikan, A., Mastrogiovanni, F., Natale, L., Kormushev, P., and Caldwell, D. G. (2015) Learning Symbolic Representations of Actions from Human Demonstrations. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, pp. 3801-3808.

Argall, B. D., Chernova, S., Veloso, M. and Browning, B., (2009), A survey of robot learning from demonstration. *Robotics and Autonomous Systems,* Vol. 57 No. 5, pp. 469-483.

Bogue, R. (2014) The role of artificial intelligence in robotics. *Industrial Robot: An International Journal*, 41(2):119-123.

Breazeal, C., Berlin, M., Brooks, A., Gray, J., Thomaz, A. L. (2006) Using perspective taking to learn from ambiguous demonstrations. Robotics and Autonomous Systems, 54(5):385-393.

Bristow, D.A., Tharayil, M. and Alleyne, A. G. (2006) A survey of iterative learning control. IEEE Control Systems Magazine, 26(3):96-114.

Buch, A.G., Kraft, D., Kamarainen, J.-K., Petersen, H.G. and Kruger, N. (2013) Pose estimation using local structure-specific shape and appearance context. *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, pp. 1050-4729.Collins, K., Palmer, A. J., and Rathmill, K. (1985) The development of a European benchmark for the comparison of assembly robot programming systems. In: *Robot technology and applications,* Berlin, Heidelberg: Springer, pp. 187-199.

Eng, T-H, Ling, Z-K, Olson W., and McLean C. (1999) Feature-based assembly modeling and sequence generation. *Computers & Industrial Engineering*, 36(1):17-33.

Fei, Y. and Zhao, X. (2005) Jamming analyses for dual peg-in-hole insertions in three dimensions. *Robotica*, 23:83-91.

Gelfond, M. and Kahl, Y. (2012) Knowledge Representation, Reasoning, and the Design of Intelligent Agents, The Answer-set programming approach. New York, NY: Cambridge University Press.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013) Dynamical movement primitives: learning attractor models for motor behaviours. *Neural computation*, *25*(2):328-373.

Jiménez, P. (2013) Survey on assembly sequencing: a combinatorial and geometrical perspective. *Journal of Intelligent Manufacturing*, 24(2):235-250.

Kormushev, P., Nenchev, D. N., Calinon, S., and Caldwell, D. G. (2011) Upper-body kinesthetic teaching of a free-standing humanoid robot. *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3970-3975.

Kuniyoshi, Y., Inaba, M., Inoue, H. (1994) Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation,* 10(6):799-822.

Krüger, N., Ude, A., Petersen, H. G., Nemec, B., Ellekilde, L.-P., Savarimuthu, T. R., Rytz, J. A., Fischer, K., Buch, A. G., Kraft, D., Mustafa, W., Aksoy, E. E., Papon, J., Kramberger, A., and Wörgötter, F. (2014) Technologies for the Fast Set-Up of Automated Assembly Processes. *Künstliche Intelligenz,* 28(4):305-313.

Lauria, S., Bugmann, G., Kyriacou, T., and Klein, E. (2002) Mobile robot programming using natural language, Robotics and Autonomous Systems 38(3-4):171-181.

Martínez, D., Alenya, G., Jimenez, P., Torras, C., Rossmann, J., Wantia, N., Aksoy E., Haller S., and Piater, J. (2014) Active learning of manipulation sequences. In: *IEEE International Conference on Robotics and Automation (ICRA),* Hong Kong, pp. 5671-5678.

Nevins, J. L. and Whitney, D. E. (1980) Assembly research, *Automatica*, 16(6):595-613.

Nicolescu, M. and Matarić, M.J. (2001) Learning and interacting in human-robot domains, IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 31(5):419-430.

Rozo, L., Jimenez, P., and Torras, C. (2013) A robot learning from demonstration framework to perform force-based manipulation tasks. *Inteligent Service Robotics*, 6:33-51.

Russell, S. J. and Norvig, P. (2010) Artificial Intelligence, A Modern Approach, Third Edition. Upper Saddle River, New Jersey: Prentice Hall.

Ramos, C., Rocha, J., and Vale, Z. (2001) A complete complexity study of one-processor assembly and manufacturing planning tasks. In: *4th IEEE International Symposium on Assembly and Task Planning*, pp 369-374.

Schreiber, G., Stemmer, A. and Bischoff, R. (2010) The fast research interface for the Kuka lightweight robot. In: *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, Anchorage, Alaska, pp. 15–21.

Ude, A., Nemec, B., Petrič T., and Morimoto, J. (2014) Orientation in Cartesian space dynamic movement primitives. In: *IEEE International Conference on Robotics and Automation (ICRA),* Hong Kong, pp. 2997-3004.

Wang, Y. and Liu, J. H. (2010) Chaotic particle swarm optimization for assembly sequence planning, *Robotics and Computer-Integrated Manufacturing*, 26(2):212-222.

Zhao, J., and Masood, S. (1999) An Intelligent Computer-Aided Assembly Process Planning System, *The International Journal of Advanced Manufacturing Technology*, 15(5):332-337.