Modular and Domain-guided Multi-robot Planning for Assembly Processes

Ludwig Nägele, Andreas Schierl, Alwin Hoffmann and Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, 86159 Augsburg, Germany

Keywords: Multi-robot Systems, Robotic Assembly, Process Planning.

Abstract: Smart factories of the future will be equipped with dynamic and task-specific teams of robots in order to manufacture custom-tailored products. For this, it is necessary to facilitate the planning of appropriate task sequences for cooperating robots. In this paper, we introduce a modular and domain-guided planning approach for multiple robots. Due to its modularity, the approach can be adapted to different assembly problems. Moreover, domain knowledge is used to guide the planning towards feasible solutions. We evaluate the approach with different examples from the blocks world domain (i. e. LEGO® DUPLO®). This evaluation shows that this domain-guided approach outperforms classical planning based on state space search such as A^*.

1 INTRODUCTION

Industrial robots offer high mechanical flexibility, speed and precision as unique features compared to other automation devices. Hence, they play an important part in the efficient automation of mass products with relatively low variability, e. g., in the automotive industry. In order to make industrial robots efficiently usable in small-lot production, a lot of research has been performed, e. g., on increasing the inherent flexibility of robots by sensor systems and associated control algorithms (Albu-Schäffer et al., 2007; Finkemeyer et al., 2010) or on new methods for efficiently bringing robots into operation (Malec et al., 2007; Andersen et al., 2015).

According to the ideas of Industry 4.0 (Kagermann et al., 2013), smart factories of the future are required to manufacture custom-tailored products with high variability in small lot sizes. As a matter of fact, the requirements of a smart factory can only be fulfilled by dynamic and task-specific cooperation of several robots (Glück et al., 2018). Building robot teams with appropriate, possibly exchangeable tools leads to multi-functional robot cells with drastically increased flexibility, performance and robustness (Angerer et al., 2015).

However, further steps have to be taken to harness the flexibility and adaptability of multiple robots for smart factories. Especially, the development of control software for the robot team working together on manufacturing a specific product must be facilitated. Hence, we present a modular and domain-guided planning approach for multiple robots in this paper. As a result of this planning, a control software for the robot team is synthesized.

The main contribution of our planning approach is that it is both modular and domain-guided. This means that the product to manufacture is analysed by domain-specific modules and is broken down into basic components and their relationship. Based on that, possible sequences of domain-specific tasks are found by the planning algorithm. In a further step, these domain tasks are used to find appropriate sequences of automation tasks based on the available robots and automation devices as well as their skills. Hence, the search space can be reduced and possible planning problems such as deadlocks can be avoided.

The paper is structured as follows: In Section 2, related planning approaches are introduced, followed by Section 3 which outlines the domain that will be used for describing and evaluating our approach. Section 4 introduces the idea of our modular planning approach, whereas the algorithms are explained in Section 5. The approach is evaluated in Section 6. Finally, a conclusion and outlook is given in Section 7.

2 RELATED WORK

Almost since its beginning, planning has been a topic of great interest in computer science. Back in 1971, at

Nägele, L., Schierl, A., Hoffmann, A. and Reif, W.
Modular and Domain-guided Multi-robot Planning for Assembly Processes.
DOI: 10.5220/0007977205950604
In *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2019)*, pages 595-604
ISBN: 978-989-758-380-3
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

SRI International the STRIPS formalism has been developed (Fikes and Nilsson, 1971), describing states of the environment and the prerequisites and results of actions. This formalism has been the base for many planning systems that work by searching the state space. To simplify transferring planning problems between different planners, the PDDL language has been developed (McDermott et al., 1998) that serves as input language to many current planners.

However, this kind of planning suffers high time complexity due to the large branching factor caused by the great amount of possible actions. To improve performance, different optimizations have been introduced. One way is to use heuristics to guide the search, such as in the A^* algorithm (Hart et al., 1968), which however require fine-tuning for certain domains. Another way has been to automatically derive levels of abstractions to be used for planning, as with ABSTRIPS (Knoblock, 1990; Giunchiglia, 1999). When working with multiple robots, the number of possible actions is vastly improved, so mechanisms using constraints and resources to guide parallel tasks have been proposed (Wilkins, 1984). However, these approaches still fail for large planning problems, while not using the abstractions that naturally exist in the planning domain that could help solve the problem in a more directed manner.

Planning is sometimes performed through task decomposition, where complex tasks are decomposed into (alternative) sequences of simpler tasks. Here, the formalism of Hierarchical Tasks Networks (HTNs) has found widespread use (Nau et al., 2003). In contrast to search-based planning, HTNs need more domain-specific knowledge but scale better for larger problems because they can use levels of abstraction present in the planning domain.

In the field of manipulation and assembly planning, different approaches have been described, including topics such as handing over objects from one robot to another (Koga and Latombe, 1994) and finding appropriate assembly sequences to build desired structures (Thomas and Wahl, 2001). Additionally, such topics have been addressed with mobile robots, where task allocation based on spatial deliberations play an important role (Stein et al., 2011; Schoen and Rus, 2013; Knepper et al., 2013).

When looking at current planning systems for multi-robot environments, task decomposition is often performed manually or through HTNs, while task allocation is performed in a search based manner (Yan et al., 2013). The same aspect can be found in the field of combined task and motion planning, where planning is performed on two levels of abstraction, mixing HTNs for task planning with search based approaches for collision-free motion planning (Wolfe et al., 2010).

This paper extends previous work of the authors (Nägele et al., 2015; Macho et al., 2016; Nägele et al., 2018) in the field of automatic assembly planning, using concepts from the aforementioned papers while making the previous solutions applicable to more complex assembly problems.

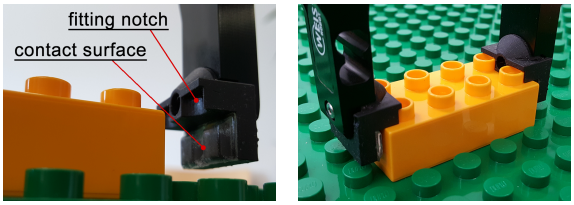


Figure 1: Specialized gripper clamps have been developed to enable process-reliable grasping of duplo bricks.

3 CASE STUDY

One main research goal of the authors is to fully automatize planning and manufacturing of large plane parts made of carbon-fibre reinforced polymers (CFRP) with industrial robots (Glück et al., 2018). However, this domain makes it hard to clearly show the contributions and benefits of the presented concepts to a wider range of non-experts. For this reason, use-cases have been developed in the well-known and properly defined blocks world domain of LEGO® DUPLO®. Here, it is possible to construct scenarios with intuitive problem definitions that show specific planning challenges. Since problem definitions are similar throughout most domains of assembly, concepts to solve duplo problems can be transferred to more complex domains of assembly such as the one of CFRP.

For the case study with duplo, a robot cell with two robots (KUKA LBR iiwa and KUKA LWR 4) is used, each equipped with a Schunk WSG 50 gripper. For reliably handling duplo bricks with these grippers, specialized clamp extensions have been devel-

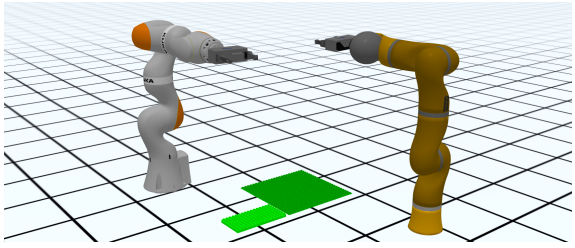
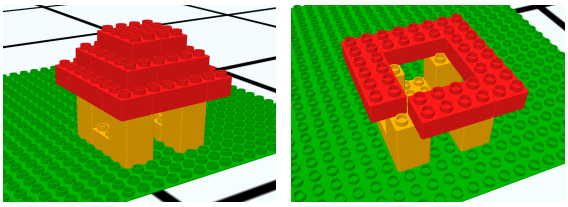
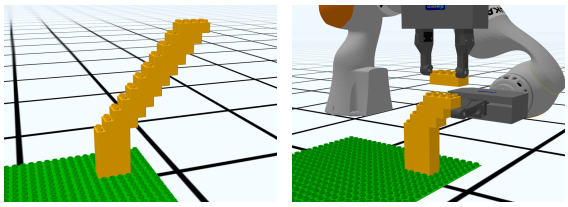


Figure 2: Virtual model of the robot cell containing two robots equipped with grippers, a duplo ground plate and an additional plate for supplying bricks.



(a) Duplo house structure (b) Deadlock scenario

Figure 3: A duplo house as an example for assembly planning (a). A missing cornerstone as cause for deadlocks (b).



(a) Duplo stairway structure (b) Supporting the structure

Figure 4: A duplo stairway (a) as an example for the need of cooperation between robots with different skills (b).

oped and manufactured using 3D printing (see Figure 1). The clamps are designed to perfectly match the contour of duplo bricks and have rubber pads acting as anti-slip contact surface. A duplo ground plate and an additional plate for supplying bricks are also part of the robot cell.

For automated planning with robots, spatial information about actuators and objects in the robot cell can be used to obtain additional data such as collisions, reachability by robots or expected movement distances (and durations) to specific targets. The virtual model of the robot cell used for the case study is shown in Figure 2. To illustratively show representative difficulties for the planning of assembly tasks, two duplo structures are used which are presented in the next two sections.

3.1 Duplo House Structure

Figure 3a shows a house made of duplo bricks, having a lower part forming a door and windows, and a roof constituted by stacked bricks aligned as a pyramid. Assuming a cornerstone not yet placed as depicted in Figure 3b, placement by a parallel gripper will result in a collision between the gripper's clamps and one of the bricks placed next to it. Grippers with another grasp strategy may however be able to place this brick though, for example by holding it with vacuum from top. Nevertheless, the available grippers cannot place this brick in the given scenario. Although further bricks can still be placed on top, it will inevitably lead to a deadlock.

Such deadlock causes, which are not yet obvious at the time they occur, can still allow a huge number of subsequent planning steps until each of them is finally stuck later on. The duplo house scenario denotes a general planning problem which is not only dependent on the current setup and the domain but also on the actuators' capabilities and their characteristics, such as their reachability. Efficient planners must address the challenge of deadlock detection in an early stage.

3.2 Duplo Stairway Structure

As a second example for planning challenges, a stairway made of duplo bricks has been designed as shown in Figure 4a. When it comes to overhanging structures, the entire structure may tend to become increasingly unstable. Not only the intrinsic stability of the structure but especially the increased force when placing a brick on top needs to be considered in order to guarantee a reliable, non-collapsing assembly.

In both cases some kind of external help is needed at a certain point. This can for example be achieved by cooperating robots: a supporting robot with its gripper clamps positioned below the brick on which the other robot places the new brick (see Figure 4b). To continue assembly, robots need to seamlessly support the stairway in a coordinated alternating manner to avoid the construction collapse while further bricks are added.

For assembly planning in general, this scenario illustrates a couple of different challenges. Besides the identification of invalid states (i. e. unstable structure), adequate helper tasks need to be scheduled at suitable points to ensure a successful assembly process. Based on that, the key task is to coordinate actuators to manage execution of both, tasks making progress as well as pure supporting tasks, and to arrange this kind of implicit cooperation. Furthermore, it has to be considered that during the time robots perform supporting tasks, they are temporarily occupied and cannot perform other tasks. All in all, this enables a huge opportunity of optimization possibilities a planner can incorporate.

4 MODULAR PLANNING

When planning the assembly of products with robots, there are mainly two steps to be considered: First, actions required to build the final artefact must be derived from a construction plan (e. g. CAD). This decides "what to do" with the single parts of the artefact (products), i. e. which abstract processes have to be

applied to the products, and possibly even which partial order of such actions is must be respected. Compared to a human who is building a duplo house, it means looking at the cover picture showing the final house and planning a mental strategy about which bricks may be placed in which order. In industrial applications, such construction decompositions are done by domain experts who know about all special characteristics and restrictions of the respective domain. Figure 5 describes such a *Domain Task* as a relationship between a *Domain Process*, which is the type of task, and a set of *Products* on which the process is applied (e. g. "Stick together" and "brick1, brick2").

So far, this process is completely independent from any robotic device yet. But in a second step, the question "how to do" all such actions in practice has to be answered. In the duplo example, it is the human searching for the next brick, grasping it with his fingers, moving it close to the underlying brick and carefully sticking them together. In industrial applications, these actions are usually performed by robots which have been programmed by process and automation experts.

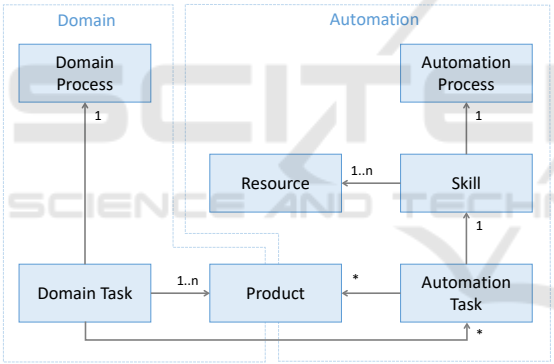


Figure 5: Overview of the different conceptual parts in both fields domain and automation.

In analogy to the domain field in Figure 5, a concrete action in the automation field performed by robots and affecting particular products is called *Automation Task*. Also here, the task is a specific implementation of a process – in this case an *Automation Processes* – applied on specific products. But since automation processes, in contrast to domain processes, are to be performed by some actuating elements (called *Resources*), an additional indirection step named *Skill* is introduced, which represents the capability of a given set of resources to perform a certain automation process. A skill, in turn, can be used to derive specific automation tasks when applied to products. A robot that can perform an automation processes "Grasp", for example, is capable of a skill "Grasp by robot" and may lead to a specific task "Grasp product x by robot". This nomenclature used for the automation field is widely based on the definition of PPRS (Pfrommer et al., 2013).

Different mechanisms are required to retrieve a valid assembly program from a construction plan. While domain, process and automation experts usually define process flow and robot programs by hand, the presented approach aims at experts transferring their knowledge into different kinds of modules instead. Such modules are meant to be independent from each other and are capable of bringing additional information about either the domain or automation into the planning process. Some are responsible for providing valid domain tasks for a construction plan, others allow to derive appropriate automation tasks for them, and others may validate the structural composition of products, for example. In the following sections, after a short introduction to modeling, three kinds of modules are presented which are used in the planning algorithm described in Section 5.

4.1 Modeling with Attributes and Situations

In recent work, the authors presented their idea of how to describe the spatial model of robots and products by *Attributes* and how this description can be derived from a construction plan (Nägele et al., 2018). In a nutshell, for the area of assembly these attributes are either a spatial position of one or a spatial relationship between several products and may also contain additional information about a specific type of relationship. An attribute describing two duplo bricks being stacked holds their position relative to each other, but also gives an idea about how the attribute needs to be performed when established: an orthogonal movement of the bricks and applying light pressure.

All attributes which are present at a specific point of time form a common set called *Situation*. Besides a goal situation, which represents the finished construction plan, each assembly process also has an initial situation, in which the assembly starts, as well as many intermediate situations during assembly.

The authors introduced *Analyzer Modules* which convert a construction plan to a corresponding goal situation (Nägele et al., 2018). The construction plan is analyzed for contained products, and their related attributes are detected, e. g. by spatial investigation. The concept of analyzer modules is a prerequisite and implicit part to the overall concept presented here.

4.2 Identifying Domain Tasks

When automatically planning the assembly of products, one element essential to the planning mechanism is knowledge about the domain. This includes a sufficient specification of domain processes, which are relevant in order to describe an entire assembly process. They are to be specified by experts of the respective domain. They are more like a unique name and do not contain any logic. From a set of available domain processes and a given goal, appropriate domain tasks are to be retrieved which are applied on specified products.

Each domain task has an impact on the previous situation on which it is performed, and finally results in a new situation being valid afterwards. This involves attributes which are established, but also attributes which are potentially removed. Since domain tasks only focus on related products but do not take robotic devices into account, their actual impact is rather idealized and inside the domain perspective.

The discovery of domain tasks is the key purpose of *domain-task identifier modules*. Implemented by domain experts, they analyse the given construction plan (in detail the difference between initial and goal situation) and identify suitable domain tasks needed for the assembly, and maybe even potential alternative. Domain-task identifier modules furthermore can determine constraints such as temporal dependencies among them. From another perspective, this gives a definition of alternatives to reach the goal with a qualified set of degrees of freedom. It is important to keep these alternatives in this stage. In a later planning step it might be necessary to fall back on alternatives if one of them does not lead to a valid result. Considering the partial order of all found domain tasks, the overall resulting attribute change must be a valid transfer from the given initial situation to the goal situation.

For the examples with duplo, one domain process "process a brick" has been specified which means as much as everything that has to be done in order to bring a brick to its target position. This may include collecting it from a magazine, pick up, move and place. Additionally, a domain-task identifier module has been developed which constructs a domain task of this type for each single brick. As domain-specific order for assembly, dependencies between the tasks are added in a way that underlying bricks are enforced to be built before.

4.3 Identifying Automation Tasks

For each identified domain task, corresponding automation tasks have to be found which map the desired behaviour of the domain task to specific automation programs. This is where *automation-task identifier modules* come into play. There are two options for experts to provide such modules:

The first type of module is a predefined mapping between a domain process and an equivalent graph of automation processes. This more or less hard-coded correlation is commonly specified by experts of both fields, domain and automation. In the duplo example, the domain process "process a brick" could be translated to a sequence of automation processes "supply a brick", "grasp a brick" and "place a brick", for example, all designated to be performed on the respective brick. For each robot capable of performing such automation processes, appropriate skills are offered by automation experts. From all skills available, automation processes are then translated to concrete tasks which can actually execute the processes. Of course, different skills may apply for the same automation processes and thus different options are eligible to choose from, such as different robotic devices or the concrete process representation. Even a skill itself may provide multiple tasks with different process parameters each. The proper selection of possible and valid variants is a challenge faced by the planning algorithm described in Section 5.

The second type of automation-task identifier modules works rather dynamically instead of using a fixed mapping. Given a domain task and the current situation in which the task is to be applied, its impact, i. e. the attributes being established or removed, can be used to derive an ideal resulting situation (goal) valid after the domain task. The same can be done with automation tasks: applied to the current situation, their resulting situation can be tested whether it approaches to the domain task's goal. This way, a sequence of automation tasks may be a valid substitute for the domain task. But even then, due to the inclusion of actuator attributes, the overall attribute change of automation tasks is more specific than the ideal one from the domain's task. Thus, when checking suitability of automation tasks for a domain task, resulting situations have to be compared without considering attributes related to any resource. For the general approach, all possible tasks have to be considered by all skills for all resources and for all products for each situation. Because a complete computation of the entire set of possibilities would be very expensive, a rather promising strategy for finding eligible and optimal tasks is preferable. Since the behavior of general automation-task identifier modules is the same for all domain tasks having no applicable fixed-mapping module, it has a default implementation as part of the planning algorithm.

4.4 Verifying Situations

During planning of assembly processes, many different tasks and situations are identified and investigated. When skills are used to provide new automation tasks, several checks regarding resources and processed products may be performed in order to reject invalid tasks, for example when a grasp target is outside of the range of a robot. However, the effect of automation tasks on raw domain-specific properties of the product-structure are not necessarily considered by all skills. While the placement of a further brick on top of a duplo stairway may be a valid automation task, the bridge itself might collapse subsequently due to its own overhanging weight. The very same task, however, would be valid in a situation where another robot is supporting an underlying brick. To prevent invalid programs and to enforce domain-specific preconditions such as implicit support assistance, automation tasks and their resulting situations need to be checked for these domain-only properties.

For this purpose, *situation verifier modules* can be contributed to the planning algorithm to validate resulting situations of tasks and to reject them where applicable. The main purpose of situation verifier modules is to enable global domain-specific checks. One or more modules can be contributed by domain experts. But also checks regarding automation can be performed by these modules, such as collision checks. An implementation of a respective situation verifier module by an automation expert makes an equivalent consideration in every single skill obsolete.

In the duplo example, a situation verifier module performs statics analysis on duplo structures in order to detect fragile constructions. Since a realistic physics simulation would vastly extend the planning time – it has to be performed on every single situation found – a rather lightweight approximation has been chosen which at least suits our case studies. The goal is to investigate all bricks from a top view and determine if their underlying support is sufficient. Straight towers are easy to identify as stable constructions, but also unobvious constructions like bridges with bricks hanging in between shall be classifiable. For this approach, the single pins of bricks are classified as stable if they are capable of carrying another brick on top. If every single brick has at least one stable pin, the entire structure is assumed to be stable. Algorithm 1 illustrates how the check is being performed. First of all, bricks are marked as fixed if they have an attribute to any base plate or a support attribute to a robot (line 3). Then, each brick laying on a cycle-free path of attributes between any two fixed bricks is also marked as fixed if it is within the convex hull of these from a top-view perspective (line 6). This enables spanning constructions like bridges. Based on the identified fixed bricks, stable pins are identified in a second step (line 9). Pins of fixed bricks are all inherent stable. For other bricks, a convex hull from top-perspective containing all stable pins of their immediately underlying bricks is spanned. All pins of the brick covered by this convex hull are also classified as stable.

Algorithm 1: Duplo stability check.

Result: Whether a structure is stable
Input : B: set containing all bricks
P: set containing all pins
$pinsOf(b \in B) \subseteq P$: pins of a brick
Output: Whether every brick has at least one stable pin

1 $F_B \subseteq B$: bricks marked as fixed;
2 $F_B \leftarrow \emptyset$;
3 **foreach** $b \in B$ **do**
4 **if** *b has attribute to baseplate* **or** *b has support attribute* **then**
5 $F_B \leftarrow F_B \cup \{b\}$;
6 **foreach** $b \in B$, $b \notin F_B$ **do**
7 **if** *b on cycle-free path between* b_1 *and* b_2, $b_1, b_2 \in F_B$ **and** *b within convex top-view hull of* b_1 *and* b_2 **then**
8 $F_B \leftarrow F_B \cup \{b\}$;
9 **Function** `stablePinsOf(`b`)`**:**
10 **if** $b \in F_B$ **then**
11 **return** $pinsOf(b)$;
12 **else**
13 $S_P \subseteq P$: stable pins of lower bricks;
14 $S_P \leftarrow \emptyset$;
15 **foreach** *lower brick* **as** b_1 **do**
16 $S_P \leftarrow S_P \cup stablePinsOf(b_1)$;
17 $hull = convexHullXY(S_P)$;
18 $pins = pinsOf(b)$;
19 **return** $pinsWithinXY(pins, hull)$;
20 **return** $\forall b \in B.\ \exists p \in stablePinsOf(b)$;

A second module discovers collisions between duplo bricks, which can mainly occur after placements. One would expect that this check is performed by the place skill itself, and that tasks with collisions are directly rejected. However, the chosen design allows to clearly divide the responsibilities of different experts, in this case an expert in maths and collision computation without knowing anything about robots or duplo.

5 MULTI-ROBOT PLANNING ALGORITHM

Using these kinds of modules, automatic planning for the corresponding domain can be performed. After analyzing the goal situation, corresponding domain tasks are identified on an abstract level. For each domain task, automation tasks are derived that reach the goal given by the domain task.

5.1 Robot Cooperation Planning

In the proposed approach, three different types of robot cooperation can occur:

(1) Two robots can explicitly cooperate by executing one automation task together, e. g. when grasping one part on two sides and cooperatively carrying it using two robots. In this case, cooperation is handled within one skill, while planning only has to make sure that in the meantime none of the robots has to perform another task.

(2) As a more implicit type of cooperation, one robot skill may depend on another robot executing or having executed a certain other task. For example, a placement operation can require another robot to support the existing structure to make sure it can withstand the pressure exerted by placing a brick. This type of cooperation becomes visible to the planning algorithm because it spans multiple unrelated tasks.

(3) Independent tasks of multiple robots can be performed at the same time, thus reducing the overall execution time. This type of cooperation has to be considered when comparing the cost for different solutions, and has to consider aspects such as collision avoidance when working in tight spaces. In this field, the implementation described in this paper uses a simple form of resource-based parallelization, however this aspect is not central to the topic of this paper and will not be explained in detail.

When performing planning with multiple robots, these aspects cause long planning times in algorithms performing state space search: on the one hand, many tasks can be performed by different robots, so that the branching factor grows with the number of capable robots. Besides, multi-step sequences can be interleaved, so that deadlocks in one multi-step sequence can only be detected after ending all parallel tasks. On the other hand, when optimizing for plan duration, actions that can be executed in parallel may not require any further time because the corresponding robot is idle at that time. In this case, many plans with the same execution duration exist to choose from.

5.2 Domain Task Planning

To overcome these specific difficulties, a two-level planning mechanism is used. First, planning is performed on the level of domain tasks (that are later resolved into multi-step sequences of automation tasks). Therefor, domain-task identification modules enumerate the domain tasks possible for a given situation and proceeding towards the global planning goal. For one of these domain tasks, the algorithm establishes an optimal automation task decomposition as described in Section 5.3. Using this domain task and automation task decomposition, search is continued. If no domain task is possible in a situation, the algorithm backtracks and continues with another domain task possible in the previous planning step.

As a further improvement towards assembly tasks (that only build up structures but do not disassemble parts), a special variant of backtracking is applied: whenever the search for an automation task decomposition of a domain task that should be possible fails, the algorithm assumes that the domain task has become impossible by one of the previously executed domain tasks (that now blocks the way). Thus, it backtracks the plan, and after each step checks if this specific domain task has become possible again, omitting all other tasks that might also be executed at that specific step. This way, the algorithm shortens the plan until either the domain task becomes possible – where it continues – or the domain task is no longer identified as possible by the domain-task identification module – then the other possible domain tasks are evaluated. This kind of backtracking helps in early detection and prevention of the corner stone deadlock seen in the house structure, where the algorithm removes bricks until the corner stone can be placed.

5.3 Automation Task Planning

After choosing a domain task, either a predefined decomposition is used, or state space search is applied to find a corresponding automation task sequence. For this, an implementation of the A^* algorithm is used, using plan execution duration and task quality as cost function. However, search is not performed in the full state space, instead only actions relevant to the chosen domain task are allowed. The corridor of relevant actions for a domain task is based on two aspects: first, the automation task has to affect at least one product mentioned in the domain task. The notion of affected products is clear for product-related tasks such as providing, picking or placing a brick, but less clear for support actions (that are defined to affect any brick above). Additionally, automation tasks affecting at-

tributes that are defined to be key of another domain task are forbidden. For the type of domain tasks used here, these mainly include placement attributes between bricks, so that valid automation tasks may not assemble or disassemble other parts of the structure.

In detail, for a situation all possible skills are asked to provide tasks with specific process parameters. For picking and placing, the skill for one robot provides individual tasks for different grasp positions. On this level, for tasks that have same result (e. g. placing a brick that has been picked up at different grasp positions) only the one with lowest cost is considered, whereas tasks that have different results (e. g. supporting a specific brick from different sides) are considered independently. This optimiziation helps to reduce the search space, while still allowing to find solutions that are only possible when using the right kind of support. Additionally, situation verifier modules are applied after each step to reject invalid situations, e. g. if the structure built is not statically stable.

Planning for a given domain task, state space search finishes once a sequence of automation tasks has been found that fulfills the requirements defined by the domain task (e. g. the chosen brick has been placed at the desired position). This sequence may establish further environment changes (e. g. robots still supporting the structure), but only if they do not interfere with other domain tasks (e. g. placing or removing another brick from the structure is forbidden).

Using this corridor of allowed actions, the planning algorithm makes sure that all planned steps belong to the task at hand, so that no irrelevant interleaving of multiple domain tasks has to be considered. This effectively reduces the search space, allows to efficiently detect impossible domain tasks, and makes it possible to handle parallel execution of independent domain tasks at a later planning (or execution) stage.

6 EXPERIMENTAL RESULTS

To evaluate the presented approach for planning assembly processes, a classical A^* solving approach is used as baseline to compare the results of the presented multi-robot planning algorithm (MRPA) with. Both case studies, the duplo house and the stairway, are planned with an A^* solver and the MRPA concepts each. The A^* implementation uses domain task and automation task options as transitions whereas situations represent states. It chooses from available domain tasks provided by the domain-task identifier module and further repeats choosing from possible automation tasks within the scope of the respective domain task to solve it. Once the goal of a domain task is reached, again all possible next domain tasks are eligible for the next step in A^*.

All situation verifier modules are used in both approaches similarly to detect invalid solutions. The cost of automation tasks is determined by their quality and in a secondary role by their execution time. For A^*, also an estimator is required which gives a guess about expected future cost from a given state. The cost of all previous tasks and the estimation is used as expected cost. For duplo, the implementation of the cost estimator is rather domain- and robotic-specific and estimates future cost by the number of bricks still to be provided or pick-and-placed and slightly underestimated average cost of respective tasks.

Both solver approaches base on cost for selection of solution paths. In case of cost equivalences, both algorithms are free which one to select next to investigate. The freedom of choice combined with random decision-making inherently bears indeterminism for the solution found as well as for the overall planning time. One evaluation running into a series of deadlocks owing to “unlucky” random decisions might be far worse than another run with only “lucky” decisions. For this reason, each case study is evaluated 100 times on each algorithm to retrieve statistically meaningful results. Once a solution is found which represents a valid result, the run is terminated. For A^* this can mean that the very optimal solution is not determined early enough in some cases, especially when cost-equal options are being evaluated. Thus, A^* is expected to finish with only a close-to-the-best solution in favor of shorter planning times, which is alright for this evaluation. This explains a possible standard deviation other than zero for the execution time of identified solutions. In general, evaluation runs lasting longer than 300 seconds usually lead to eventual heap memory problems without returning valid results. These runs are canceled and not considered in the statistical results.

To satisfy the multi-robot purpose, the robot cell as described in Section 3 is used for the evaluation. All evaluation runs, completely implemented in Java, have been executed on a Windows machine with an Intel(R) Core(TM) i7-7600U (2.80GHz) and 24 GB RAM. Table 1 gives a summary of all measured values, each with an average value, min and max occurrences, and a computed standard deviation.

6.1 Deadlock Detection And Prevention

The purpose of the duplo house case study is to illustrate deadlocks which can not be identified in the moment they arise. Additionally, a light form of multi-robot cooperation is addressed with assign-

Table 1: Evaluation runs for duplo house and stairway, each applied to A^* and the multi-robot planning algorithm (MRPA).

		Duplo house		Duplo stairway	
		A^*	MRPA	A^*	MRPA
Evaluation runs	Successful	73	100	100	100
	Timeout (>5min)	27	0	0	0
Planning time	Average	3.957 s	0.314 s	7.653 s	0.332 s
	Min - Max	0.299 - 81.004 s	0.204 - 1.361 s	7.340 - 8.685 s	0.270 - 0.671 s
	Std. deviation	12,425 s	0.101 s	0.101 s	0.000 s
Investigated automation tasks	Average	1109	209	16840	610
	Min - Max	246 - 21642	179 - 567	16810 - 16912	610 - 610
	Std. deviation	3368	47	19	0
Observed unique situations	Average	533	62	2013	66
	Min - Max	59 - 10822	52 - 217	2010 - 2017	66 - 66
	Std. deviation	1659	20	1	0
Solution length (automation tasks)	Average	52	52	45	45
	Min - Max	52 - 52	52 - 52	45 - 45	45 - 45
	Std. deviation	0	0	0	0
Solution execution time	Average	107.000 s	106.722 s	114.679 s	115.624 s
	Min - Max	104.533 - 109.536 s	103.843 - 109.998 s	114.472 - 114.861 s	115.423 - 115.810 s
	Std. deviation	0.993 s	0.980 s	0.000 s	0.000 s

ment of tasks and coordination of exclusively allocatable workspaces and thus basic collision avoidance. Collision-free motion planning for transfer tasks is not considered in this stage. Table 1 shows the results of planning the house with A^*. 27% of all runs did not terminate within the first 300 seconds because they run into the cornerstone-deadlock problem too often and waste time with planning on wrong premises. The other 73% had a fairly good planning time of almost 4 seconds. However, the random occurrence of time-waste within deadlocks causes a high standard deviation of more than 12 seconds. Compared to this, MRPA states a better average planning time and a far smaller standard deviation of it due to the detection and prevention of deadlocks. Both approaches result a solution with the same amount of automation tasks. Although MRPA seems to find a better solution with a shorter execution time compared to A^*, the solution may contain more sidewards offset grasps of bricks which is considered to cause bad cost.

6.2 Multi-robot Cooperation

In order to assemble the duplo stairway, a robot is needed supporting before a brick can be placed. This use case mainly exemplifies the capability of composing independent skills to perform tasks neither of the skills could execute solely. One difficulty here is that selecting a support task does not involve an immediate profit in fitness. Only in a later state, e. g. when a placement is now possible, the profit appears. This results in a kind of temporary blind search for both, A^* and MRPA. Since MRPA does not optimize globally (on domain task layer), such cooperations by independent skills might not be solved optimal. While A^* finds solutions with better execution times for the stairway, MRPA however investigates far less automation tasks and situations and thus saves planning time. Because all domain tasks need to be selected sooner or later, a prioritization of cheap ones and thus the finding of slightly better solutions might not seem worth the far worse planning time and thus is left out in MRPA. However, rare cases exist where omitting global optimization may lead to significantly worse solution results. Finally, a support is not needed in all cases, which is however unknown in advance. Such uncertainties make cost estimation really difficult and especially the development of corresponding modules. As A^* is based on cost estimation, this forms a disadvantage compared to MRPA.

7 CONCLUSION

In this work, an approach for modular and domain-guided multi-robot planning for assembly processes has been presented. Modular concepts are introduced for separately contributing expert knowledge from different areas such as domain and automation. General planning difficulties regarding coping with deadlocks and planning multi-robot tasks are identified and respective possible algorithmic solving strategies are proposed. Based on these results, two examples in the domain of LEGO® DUPLO® containing the identified difficulties are introduced: one containing several situations eventually leading to deadlocks, and one requiring implicit cooperation of two robots. The proposed multi-robot planning algorithm (MRPA) is

applied to both, and the results are evaluated statistically and compared to the results of a classical A^* implementation. MRPA performs well with the modular concepts and finds good solutions without running into huge state-space explosions. However, it is not yet capable of global optimization, which will be addressed in ongoing work. Besides, also improved handling of robot cooperation based on independent skills is planned to be investigated in future research.

ACKNOWLEDGEMENTS

This work is partly funded by the German Research Foundation (DFG) under the TeamBotS grant.

REFERENCES

Albu-Schäffer, A., Ott, C., and Hirzinger, G. (2007). A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *Intl. J. Rob. Research*, 26(1):23–39.

Andersen, R. H., Dalgaard, L., Beck, A. B., and Hallam, J. (2015). An architecture for efficient reuse in flexible production scenarios. In *2015 IEEE Conf. on Autom. Science and Engineering*, pages 151–157.

Angerer, A., Vistein, M., Hoffmann, A., Reif, W., Krebs, F., and Schönheits, M. (2015). Towards multi-functional robot-based automation systems. In *12th Intl. Conf. on Inform. in Control, Autom. & Robot.*, pages 438–443.

Fikes, R. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208.

Finkemeyer, B., Kröger, T., and Wahl, F. (2010). The adaptive selection matrix - a key component for sensor-based control of robotic manipulators. In *2010 IEEE Intl. Conf. on Robotics and Autom.*, pages 3855–3862.

Giunchiglia, F. (1999). Using abstrips abstractions – where do we stand? *Artificial Intelligence Review*, 13(3):201–213.

Glück, R., Hoffmann, A., Nägele, L., Schierl, A., Reif, W., and Voggenreiter, H. (2018). Towards a tool-based methodology for developing software for dynamic robot teams. In *15th Intl. Conf. on Inform. in Control, Autom. & Robot.*, pages 615–622.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4(2):100–107.

Kagermann, H., Wahlster, W., and Helbig, J., editors (2013). *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. acatech.

Knepper, R. A., Layton, T., Romanishin, J., and Rus, D. (2013). Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE Intl. Conf. on Robot. & Autom.*, pages 855–862.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *8th Nat. Conf. on Artificial Intelligence*, pages 923–928. AAAI Press.

Koga, Y. and Latombe, J. . (1994). On multi-arm manipulation planning. In *1994 IEEE Intl. Conf. on Robot. & Autom.*, pages 945–952.

Macho, M., Nägele, L., Hoffmann, A., Angerer, A., and Reif, W. (2016). A flexible architecture for automatically generating robot applications based on expert knowledge. In *47th Intl. Symp. on Robotics*.

Malec, J., Nilsson, A., Nilsson, K., and Nowaczyk, S. (2007). Knowledge-based reconfiguration of automation systems. In *2007 IEEE Intl. Conf. on Autom. Science and Engineering*, pages 170–175.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL — the planning domain definition language. Technical Report TR 98 003/DCS TR 1165, Yale Center for Computational Vision and Control.

Nägele, L., Macho, M., Angerer, A., Hoffmann, A., Vistein, M., Schönheits, M., and Reif, W. (2015). A backward-oriented approach for offline programming of complex manufacturing tasks. In *6th Intl. Conf. on Autom., Robotics & Applications*, pages 124–130. IEEE.

Nägele, L., Schierl, A., Hoffmann, A., and Reif, W. (2018). Automatic planning of manufacturing processes using spatial construction plan analysis and extensible heuristic search. In *15th Intl. Conf. on Inform. in Control, Autom. and Robot.*, pages 576–583.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). Shop2: An htn planning system. *J. Artif. Int. Res.*, 20(1):379–404.

Pfrommer, J., Schleipen, M., and Beyerer, J. (2013). PPRS: production skills and their relation to product, process, and resource. In *18th IEEE Conf. on Emerging Techn. & Factory Autom.* IEEE.

Schoen, T. R. and Rus, D. (2013). Decentralized robotic assembly with physical ordering and timing constraints. In *2013 IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, pages 5764–5771.

Stein, D., Schoen, T. R., and Rus, D. (2011). Constraint-aware coordinated construction of generic structures. In *2011 IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, pages 4803–4810.

Thomas, U. and Wahl, F. M. (2001). A system for automatic planning, evaluation and execution of assembly sequences for industrial robots. In *2001 IEEE/RSJ IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, pages 1458–1464.

Wilkins, D. E. (1984). Domain-independent planning representation and plan generation. *Artificial Intelligence*, 22(3):269–301.

Wolfe, J., Marthi, B., and Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *20th Intl. Conf. on Automated Planning and Scheduling*, pages 254–257. AAAI Press.

Yan, Z., Jouandeau, N., and Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *Intl. J. of Advanced Robotic Systems*, 10(12):399.