2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

LegoBot: Automated Planning for Coordinated Multi-Robot Assembly of LEGO structures*

Ludwig Nägele, Alwin Hoffmann, Andreas Schierl and Wolfgang Reif

Abstract— **Multi-functional cells with cooperating teams of robots promise to be flexible, robust, and efficient and, thus, are a key to future factories. However, their programming is tedious and AI-based planning for multiple robots is computationally expensive. In this work, we present a modular and efficient two-layer planning approach for multi-robot assembly. The goal is to generate the program for coordinated teams of robots from an (enriched) 3D model of the target assembly. Although the approach is both motivated and evaluated with LEGO, which is a challenging variant of blocks world, the approach can be customized to different kinds of assembly domains.**

I. Introduction

LEGO is a construction toy with hundreds of different pieces leading to a high variability of possible structures. The main components of every structure are LEGO bricks of different sizes. A brick is a cuboid with knobs on the top in a given grid and precisely fitting counterparts on the bottom. Bricks can be stacked within this grid using right-angled rotations. As putting two LEGO bricks together can be seen as a joining process [1], the building of LEGO structures is a perfect example for assembly. Due to its combinatorial complexity, the LEGO domain is a suitable representative for automatically planning robotic assembly sequences and for evaluating different approaches and algorithms.

In contrast to common blocks world examples for planning [2], LEGO offers additional challenges. Fig. 1a shows a situation in which a LEGO brick can no longer be placed in its target position due to the knobs. This *exclusion* problem requires to dismantle the structure partially in order to place the missing brick. A related problem occurs when a LEGO brick cannot be placed due to a *collision* between neighboured bricks and the gripper holding this brick (cf. Fig. 1b). In both situations, a planner can still explore a huge set of further states without detecting this deadlock situation. This requires appropriate strategies for backtracking at an early stage in order to efficiently find a plan.

If LEGO bricks are assembled in order to form a staircase, the structure will collapse from a certain height due to its own weight. The staircase depicted in Fig. 1c is only stable if its upper part is supported, e. g., by the other structure shown. If you imagine a manual assembly of such a staircase, every placement of a brick would have to be supported by a second hand until the overall construction reaches a stable condition. Hence, the *statics* of LEGO structures must be

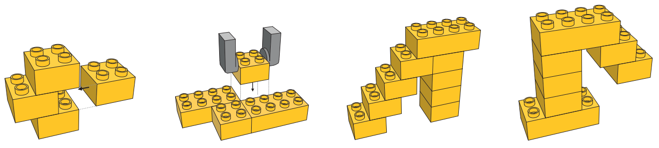

(a) Exclusion (b) Collision (c) Statics (d) Overhang

Fig. 1: Planning challenges inherent to the LEGO domain in contrast to other common blocks world examples.

regarded during planning. Another challenge, the *overhang* problem, is shown in Fig. 1d. Here, a straightforward bottom-up construction would not lead to a valid plan, since the bricks of the overhanging substructure can only be attached towards the end, although they belong to lower levels. That is why a simple bottom-up strategy – level by level – is not feasible in general. Instead, a planning approach is required that is independent of the spatial position of LEGO bricks.

In this paper, we propose an AI-based planning approach and its formal description for coordinated multi-robot assembly. Forming coordinated teams of robots leads to multi-functional robot cells with drastically increased flexibility, performance and robustness [3] which are key requirements of future factories according to the paradigm shift of Industry 4.0. Even though our approach is illustrated and evaluated with LEGO structures, we are confident that it is a general approach for assembly with coordinated teams of robots due to its modular organization which allows for customising to different domains. The LEGO example help us to isolate genuine planning issues, and to separate them from domain-specific complications.

A systematic approach [4] is essential as the programming, coordination and scheduling of robot teams is a tedious and complex task. Hence, the main contributions of the paper are

1) the introduction of our modular, two-layer planning approach for multi-robot assembly in Sect. III,
2) an associated formal description of the planning domain (cf. Sect. IV),
3) a planning algorithm (cf. Sect. V) based on the formal description, and
4) an evaluation in Sect. VI based on two different LEGO constructions.

For the experimental setup (cf. Sect. II), we have chosen to use LEGO DUPLO, which has larger bricks compared to classical LEGO and, thus, facilitates handling with industrial robots and grippers. To show the advantages of our approaches, related work is presented in Sect. VII. Finally, the paper concludes with Sect. VIII by giving a discussion.

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

The authors are with the Institute for Software & Systems Engineering, University of Augsburg, Universitätsstr. 6a, Augsburg, Germany. ludwig.naegele@informatik.uni-augsburg.de

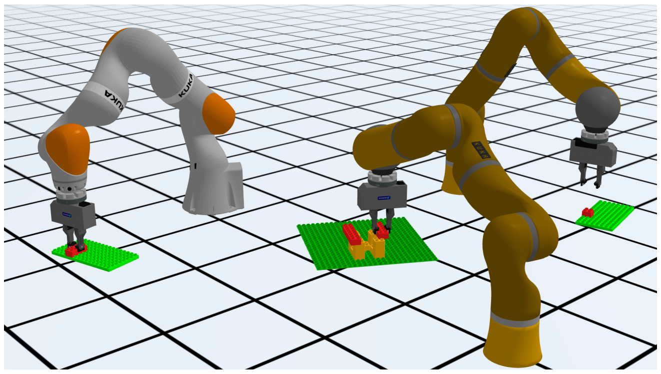
II. Experimental Setup

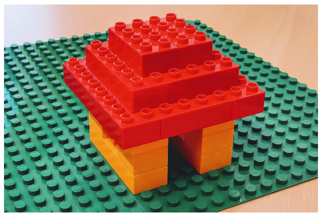
To automatically perform planning steps like allocating appropriate robots based on reachability analysis and collision-free motion planning, spatial information about robots and grippers as well as LEGO bricks and structures is needed. A virtual model representing the real robot cell contributes such information (see Fig. 2a). For modelling the robot cell with its devices, we use the Robotics API [5] which allows for reasoning about geometric and semantic information [6]. Moreover, the Robotics API is able to simulate even coordinated motions of multiple robots during planning and is subsequently able to execute these motions with real-time guarantees [7]. Hence, we have a kinematic digital twin with robots, grippers and LEGO bricks as well as their semantically enriched relations available during planning.

For the DUPLO case studies, the initial robot cell (see Fig. 2a) consists of a large DUPLO ground plate which is the foundation of every structure to be built. The robots, which will be used for planning and assembly, are arranged around this ground plate. For our experiments, we are using up to three lightweight robots each with a parallel gripper. Due to special clamps with a negative brick profile, the grippers can reliably handle DUPLO bricks. Moreover, each robot is also equipped with a dedicated smaller DUPLO plate for supplying bricks. The position of the robots with respect to the ground plate largely influences the planning as the kinematic digital twin verifies reachability. Hence, there will be no solution, if the setup of the robot cell is unfavourable.

The first of two LEGO case studies is the DUPLO house which is depicted in Fig. 2b. The house has an area of $128\,\mathrm{mm} \times 128\,\mathrm{mm}$ and a height of approximately $116\,\mathrm{mm}$. It consists of a lower base part with one door and three window sides and a pyramid like roof top. The structure is built on a ground plate with 26 bricks on 6 levels: 4 red and 9 yellow bricks with a grid of 2×2 knobs as well as 10 red and 3 yellow bricks with 4×2 knobs are used. The house can be constructed using one or more robots and contains multiple deadlock situations on the corner bricks of the roof top due to the collision problem depicted in Fig. 1b. Such a deadlock situation is not detectable at the time it originally occurs, because there are still many permissible subsequent steps. This leads to high planning times with exponential complexity subject to the number of bricks and skills. Even this rather simple example might not be solvable in adequate time when having to examine all these possibilities.

The second LEGO case study is the DUPLO bridge which is depicted in Fig. 2c. The arch bridge spans approximately $290\,\mathrm{mm}$ and is built on a ground plate with 64 LEGO bricks on 14 levels: 32 yellow bricks with a grid of 2×2 knobs, 22 yellow bricks with 4×2 knobs, one yellow brick with 10×2 knobs as well as 6 yellow bricks with 2×1 knobs and double height are required for the arch. For the road on top, 3 grey bricks with a grid of 8×2 knobs are necessary. This case study was chosen because it can only be assembled with at least three robots and it shows all the problems of the LEGO domain introduced in Fig. 1. If the arch bridge


(a) Simulated robot cell for assembling LEGO structures


(b) House

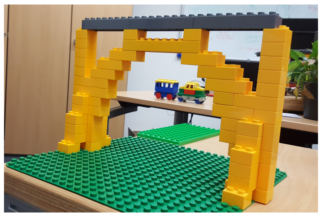

(c) Bridge

Fig. 2: The coordinated multi-robot assembly uses up to three lightweight robots grouped around a DUPLO ground plate (a) in order to build the two DUPLO case studies (b), (c).

is built from one side, the remaining stones on the other side of the arch can be added from the top brick to the bottom and, thus, can be viewed as an overhang. However, this overhang situation would also lead to an exclusion, since at least the last brick between the almost finished arch and the base plate can no longer be inserted. Moreover, the exclusion problem can occur at both abutments of the arch. Below the imposts, there are a large number of possible sequences to place the LEGO bricks in order to form both pillars, which lead to collisions between gripper and bricks. In general, an arch bridge is a self-supporting structure which transfers its weight and its loads partially into a horizontal thrust restrained by abutments at both sides. However, if you remove any brick from the arch, the self-supporting property is no longer given and the bridge would collapse. As a consequence, both sides of the arch become unstable at a certain height during construction and must be somehow supported as long as the uppermost brick is missing. That is also why the bridge can only be constructed with at least three robots: two robots are required for supporting both sides of the arch and one for placing the uppermost brick. Hence, the statics problem is apparent for the LEGO bridge and must be coped with in the planning approach.

III. Planning Approach

Fig. 3 shows a schematic overview of the planning approach which is divided into three different phases: (A) structural analysis, (B) process planning, and (C) production planning. Whereas the first two phases are domain-specific in order to decompose the problem and find a feasible task sequence, the last phase is responsible for finding

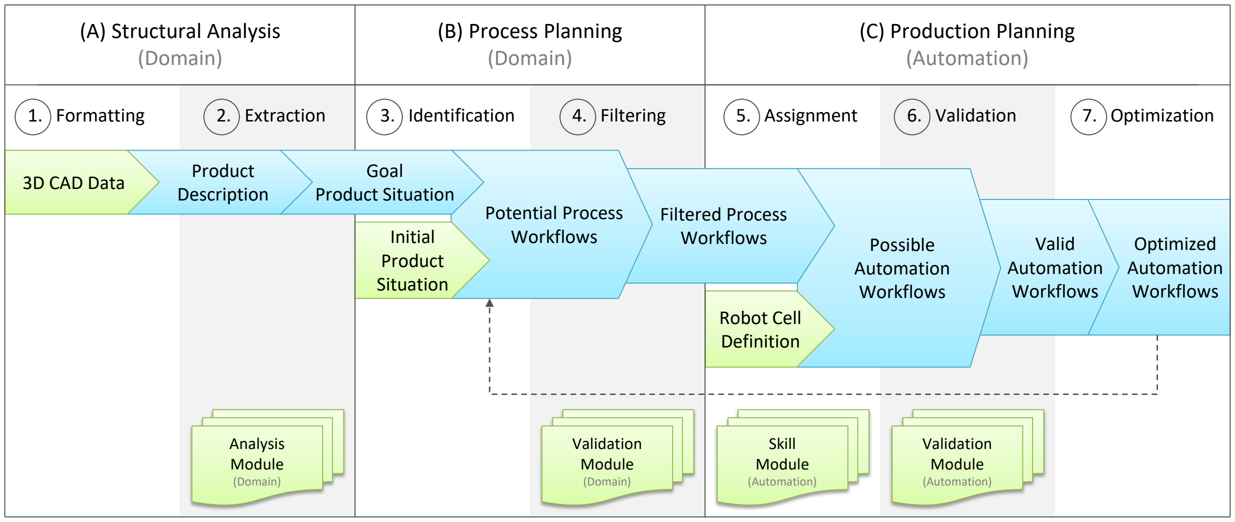


Fig. 3: Overview of the presented planning approach which is divided into three different phases. The main idea is to distinguish between domain and automation planning, but still to interweave both parts in an appropriate way.

suitable robotic devices which are capable to manufacture the product. The customisation to the domain and to specific automation devices is achieved by expert modules contributing to the planning at defined points (marked as green boxes in Fig. 3). In this way, expert domain as well as expert automation knowledge can be fed into planning.

In phase A, i. e., structural analysis, the model of the product to be manufactured is converted into a format suitable for planning [8]. The formatting step (1) is used to translate the product model, which is often available as 3D CAD data or in a proprietary data format, into a uniform *product description* that spatially represents the overall construction. In a second step, the extraction (2) identifies existing relationships (e. g., joining processes, mechanical fasteners) between the various components of the assembly and converts them into a representation for planning, the so-called *goal product situation*. In our LEGO example, the description of a construction is usually available as a set of pictures. The used DUPLO bricks are identified during the formatting step and their spatial position is determined. During extraction, the spatial information is used to determine which relationships between individual bricks must be present in the overall construction (i. e., which knobs must be connected to which bricks).

In phase B, i. e., process planning, an initial planning takes place at an abstract domain-specific level without considering the available robots and tools [9]. The identification step (3) determines possible domain tasks for establishing the aforementioned relationships and concatenates them to *potential process workflows*. During this step, the *initial product situation* is required in addition to the desired goal product situation in order to derive possible domain tasks based on the differences of both. Due to a filtering step (4), invalid process workflows are excluded from the planning at an early stage in order to simplify further planning. For LEGO, the process planning identifies possible sequences in which the bricks can be put together for establishing the overall construction in an appropriate way. Hence, sequences in which exclusions occur are discarded.

Phase C, i. e., production planning, is dedicated to generate a specific automated solution for previously selected process workflows with the robots and tools available [9]. During the assignment step (5), a given *robot cell definition* is used to calculate how the individual steps of a process workflow can be realised with the available skills of the robot cell. For the parametrisation of skills, techniques for including expert knowledge [10], [11], programming by demonstration [12], [13] or constraint-based programming [14], [15] can be applied. The resulting *possible automation workflows* are examined in a subsequent validation step (6) in order to filter invalid workflows, e. g., with colliding robots or impossible intermediate product situations. Because the remaining *valid automation workflows* contain only sequences of tasks, their overall execution time can be reduced by using parallel scheduled robots for assembly. This is done in the optimisation step (7) to obtain the best possible planning result. In our LEGO example, abstract assembly steps (e. g., “process the next brick”) are transferred to automation sequences, e. g., by grasping, transporting and placing a brick with a specific robot and gripper. However, this can lead to collisions or unstable intermediate structures (e. g., a partially built bridge without necessary support) which are filtered out.

In general, there are countless options to determine a specific automation workflow from a given product description. Already in the identification step (3), there are usually many possibilities how the product can be assembled from an abstract point of view. Each possible process workflow consists of a sequence of domain tasks, for which there is a large number of possible automation solutions in the assignment step (5), depending on the robot cell itself. If you combine all possible solutions of each domain task with all possible process workflows, it will lead to an immense solution space, in which the search for an optimal solution

– maybe even for an existing solution – is computationally expensive. An examination of all possible workflows – both in process and in production planning – is therefore not feasible, especially for complex planning problems.

Instead, our planning approach follows an interwoven, incremental strategy (indicated by the dashed arrow in Fig. 3) that pursues a two-layer planning procedure with a macro and micro step division between process and production planning [9], [16]. During process planning, a first process step is determined that brings the initial product situation closer to the goal product situation. Following the idea of a search corridor, only for this domain task, an automation solution (i. e., a sequence of automation tasks) that is valid on the one hand and as optimal as possible on the other hand is calculated during the subsequent production planning in steps (5) – (7). Once an optimal automation solution for this domain task has been determined, process planning starts again at step (3) and, based on the previous result, is continued with the selection of the next domain task to be examined. If, however, no automation solution can be found for a domain task, an alternative is searched for at the level of process planning. This procedure is repeated until a workflow is found that describes the complete assembly of the desired goal product situation.

IV. Formal Definition

For describing planning states and planning progress, a concept named *Attribute* is used. An attribute a is given as predicate that indicates whether a relationship or a property of the product is valid (constructed) at a given stage of assembly. For example, an attribute might express a stacking of two DUPLO bricks with a specified geometric offset. It is distinguished between attributes which reference solely assembly parts of the product, which are called *product attributes*, and attributes which also relate robotic devices, as for example the grasp of a part by the gripper. These are called *actuator attributes*. The set of all attributes is called $\mathcal{A}$, while $\mathcal{A}\restriction Product \subseteq \mathcal{A}$ denotes all product attributes and $\mathcal{A}\restriction Actuator = \mathcal{A} \setminus \mathcal{A}\restriction Product$ all actuator attributes. For subsets of attributes, a predicate $cons : \wp\mathcal{A} \rightarrow \mathbb{B}$ expresses *consistency*. A set is called consistent, if and only if all of its attributes can coexist in real without contradictions.

For planning on both macro and micro level (i. e. process and production planning), two different state representations are required. A *planning situation* $s \subseteq \mathcal{A}$, on the one hand, describes a consistent set of attributes (i. e., $cons(s)$) that are constructed at a given stage of assembly. The set of all situations is called $\mathcal{S}$. On the other hand, a set of product attributes is given as *product situation* $p \subseteq \mathcal{A}\restriction Product$ if it describes a certain product state during planning. For a planning situation s, its corresponding product situation is defined as $s\restriction Product$. It is $\mathcal{S}\restriction Product = \{s\restriction Product \mid s \in \mathcal{S}\}$ the set of all product situations. An example of a LEGO product state described by a product situation is given in Fig. 4. Four valid product attributes (details see below) establish the structure, each connecting two bricks.

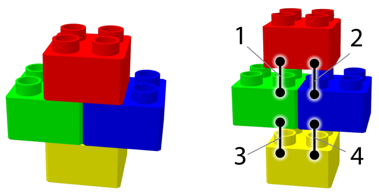


Fig. 4: LEGO structure described by a product situation

The overall aim of assembly planning is to find robotic tasks which result in a planning situation that represents the completely assembled product according to a goal product situation. In this final planning situation, the specific constellations of the actuators do not matter to the reached goal. Instead, only product related parts of the planning situation count for the assembly process. A planning situation $s \in \mathcal{S}$ *fulfils* a product situation $p \in \mathcal{S}\restriction Product$, if and only if s and p contain the same product attributes; which is $s \Vdash p$ exactly if $s\restriction Product = p$.

Let t be a *Task* and $\mathcal{T}$ the set of all tasks. Attributes, which are explicitly named by a task $t \in \mathcal{T}$ to be constructed after its execution, are called $ExplEffect^{+}(t) \subseteq \mathcal{A}$. In turn, $ExplEffect^{-}(t) \subseteq \mathcal{A}$ denotes the set of attributes which are explicitly named to be removed after the task's execution. For each task t, $ExplEffect^{+}(t) \cap ExplEffect^{-}(t) = \emptyset$ holds. Thus, the pair

$$ExplEffect(t) = (ExplEffect^{+}(t), ExplEffect^{-}(t))$$

completely describes the defined attribute change that takes place after execution of t. For the planning on macro level, a special form of task is introduced: The set of *domain tasks* t is defined as $\mathcal{T}\restriction Product$ so that $ExplEffect^{+}(t) \subseteq \mathcal{A}\restriction Product$ and $ExplEffect^{-}(t) \subseteq \mathcal{A}\restriction Product$. That is, domain tasks do only construct or remove product attributes. A task $t \in \mathcal{T} \setminus \mathcal{T}\restriction Product$ that also influences actuator attributes is called *automation task*.

Besides the attributes explicitly defined by a task, other attributes may also be affected by a task's execution. Assume a task explicitly constructing attribute 2 in Fig. 4. Having attributes 3 and 4 already assembled before, also attribute 1 is constructed by the task. In a product situation where none of the attributes exists before, however, only attribute 2 is constructed by the same task. Which attributes actually are modified depends on the situation the task is executed in. Given a task $t \in \mathcal{T}$ and a situation $s \in \mathcal{S}$, $Effect^{+}(t,s)$ and $Effect^{-}(t,s)$ describe disjoint sets of attributes that are actually constructed or removed by t in s, with: $ExplEffect^{+}(t) \subseteq Effect^{+}(t,s)$ and $ExplEffect^{-}(t) \subseteq Effect^{-}(t,s)$. Analogously, the pair

$$Effect(t,s) = (Effect^{+}(t,s), Effect^{-}(t,s))$$

completely describes the actual attribute change that takes place after execution of t in s. The resulting situation $next(t,s)$ after execution of t in s is given by

$$next(t,s) = (s \setminus Effect^{-}(t,s)) \cup Effect^{+}(t,s) \ .$$

Knowing all attributes that are constructed explicitly and those that are actually constructed, the pair

$$CollEffect(t, s) = (CollEffect^{+}(t, s), CollEffect^{-}(t, s))$$

with $CollEffect^{+}(t, s) = Effect^{+}(t, s) \setminus ExplEffect^{+}(t)$ and $CollEffect^{-}(t, s) = Effect^{-}(t, s) \setminus ExplEffect^{-}(t)$ names all those attributes that are *collaterally* constructed or removed by task t in situation s. With all this at hand, the real effects of tasks can be considered during planning.

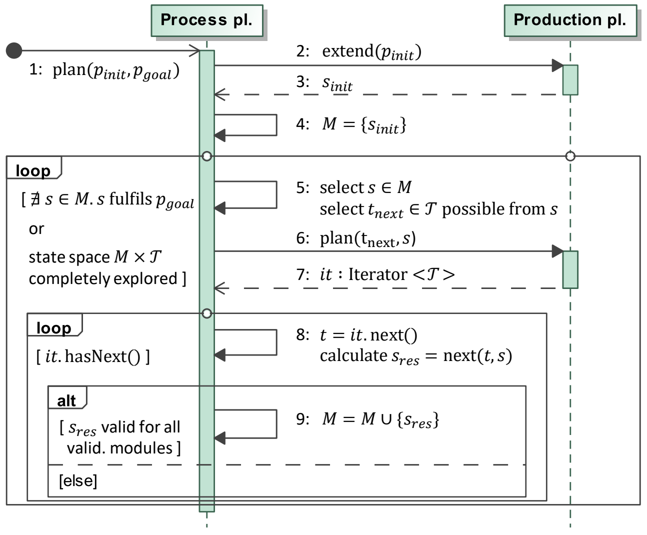
V. Planning Algorithm

Whether a planning problem can be solved, depends, amongst others, on the available actions that constitute the state space for the search. For more complex problems, where the state space may be huge and only certain paths lead to valid goal states (e. g., the corner bricks of the house that cause collision problems), classical search strategies such as depth-first search and breadth-first search fail with given resources and time, while others like A* require at least highly specialised heuristics [17]. Our proposed two-layer planning strategy for multi-robots (Multi-Robot Planning Algorithm, MRPA) is based on macro and micro steps and aims at outperforming classical strategies due to optimisation techniques assumed valid for general assembly planning. On the macro step level, i. e., process planning, a sequence of abstract domain tasks is searched which are planned with automation tasks each on the micro step level, i. e., production planning.

Fig. 5 describes the strategy for searching the state space where planning situations are used as states and tasks as transitions. Given an initial product situation $p_{init} \in \mathcal{S}\restriction Product$, in which assembly starts, and a goal product situation $p_{goal} \in \mathcal{S}\restriction Product$, which describes the final artefact (1), first a respective planning situation $s_{init} \in \mathcal{S}$ with $p_{init} = s_{init}\restriction Product$ is retrieved from the production planning layer (2, 3) which also describes the robot cell on assembly start. s_{init} is added to the set M of states which have to be further investigated (4). From this set of known planning situations, one is selected as next (it is s_{init} in the first run) and all its possible next domain tasks (e. g., assemble a brick) as macro steps are determined. From those, one domain task is selected as next abstract task t_{next} (5). t_{next} and situation s, in which t_{next} is performed, are given to the production planning layer which performs a local A* on micro step level with adapted search scope in order to provide executable automation tasks – maybe hierarchically nested – for the abstract domain task (6, 7). Technically, the result is returned as iterator that performs production planning only if $next()$ is invoked and provides all alternatives for the local assembly.

In order to determine resulting planning situations during production planning, the determination of collateral effects is used. A solution t is returned, if and only if its resulting planning situation $s_{res} = next(t, s)$ (8) fulfils the requirements of t_{next}, i. e.,

$$s_{res} \Vdash (s\restriction Product \setminus Effect^{-}(t_{next})) \cup Effect^{+}(t_{next}) \ .$$



Process pl.
Production pl.
1: plan(p_{init}, p_{goal})
2: extend(p_{init})
3: s_{init}
4: $M = \{s_{init}\}$
loop
[$\nexists\, s \in M.\ s$ fulfils p_{goal}
or
state space $M \times \mathcal{T}$
completely explored]
5: select $s \in M$
select $t_{next} \in \mathcal{T}$ possible from s
6: plan(t_{next}, s)
7: it : Iterator <$\mathcal{T}$>
loop
[it.hasNext()]
8: $t = it$.next()
calculate s_{res} = next(t, s)
alt
[s_{res} valid for all
valid. modules]
9: $M = M \cup \{s_{res}\}$
[else]

Fig. 5: Algorithm description with macro and micro layer.

Each such t is – if approved to be valid by all validation modules – handled as potential next step to the overall solution and its resulting planning situation s_{res} is added to the set M (9), from where it is selectable as new initial state for a further step of assembly planning. As long as no $s \in M$ fulfils the goal state $s \Vdash p_{goal}$, i. e., all product attributes of the final product are constructed in s, planning is continued.

Key to the success of the MRPA is also the specific strategy of selecting the next state for planning as it directly influences the overall performance of the search. Assuming that the chosen order of assembly steps on macro level (5) does not play a major role to the overall performance of the final assembly, the selection of next domain tasks is done in a depth-first like manner, where special backtracking strategies apply for avoiding exhaustive planning due to deadlocks [9]. However, the selected domain task sets a corridor for further planning at micro step level. Here, the local A* search for finding suitable automation tasks is restricted to skill modules that only affect the respective assembly part. The number of combinational possibilities is thus drastically reduced. Assuming n as the number of parts that can be assembled independently (i. e., domain tasks) and k as the number of actions (i. e., automation tasks needed in order to fulfil such

TABLE I: Size of state-space with n as number of parts to be assembled and k as number of actions needed per part.

	n	1		2		3		4		5	
k		full	corr.	full	corr.	full	corr.	full	corr.	full	corr.
1	states	1	1	4	6	15	27	64	124	325	645
	paths	1	1	2	2	6	6	24	24	120	120
2	states	2	2	18	12	270	54	7,363	248	326,010	1,290
	paths	1	1	6	2	90	6	2,520	24	113,400	120
3	states	3	3	68	18	5,247	81	1,107,696	372	492,911,195	1,935
	paths	1	1	20	2	1,680	6	369,600	24	168,168,000	120

TABLE II: Evaluation runs for LEGO house and bridge, each solved with classical A^* and the multi-robot planning algorithm.

Problem:	**LEGO house**				**LEGO bridge**	
Solver:	A^*		MRPA		A^*	MRPA
# Robots:	1 robot	3 robots	1 robot	3 robots	3 robots	3 robots
Successful runs:	250 / 250	167 / 250	250 / 250	250 / 250	1 / 250	250 / 250
Planning time:	2.17 s ± 3.34 s	15.20 s ± 35.53 s	0.39 s ± 0.32 s	1.98 s ± 1.76 s	277.36 s ± N/A	64.84 s ± 44.94 s
Domain tasks:	52 ± 0	52 ± 0	52 ± 0	52 ± 0	141 ± N/A	141.2 ± 1.1
Automation tasks:	338 ± 0	338 ± 0	338 ± 0	338 ± 0	872 ± N/A	873.06 ± 3.20
Execution time:	230.39 s ± 0.15 s	174.42 s ± 16.59 s	230.39 s ± 0.15 s	90.14 s ± 3.51s	489.47 s ± N/A	431.98 s ± 32.41 s

a domain task). Table I contains the number of states and paths available in the state-space for both, a full and a corridor-guided search. Even for a small planning problem with $n = 5$ parts and $k = 3$ steps per part, almost half of a billion states span the search-space. In contrast, the corridor approach requires only about 2,000 states for planning. It can be observed that states increase only linearly with the number of actions k for the corridor approach in this example. Noticeably decreasing the state-space, this forms a major benefit to planning performance.

VI. Experimental Results

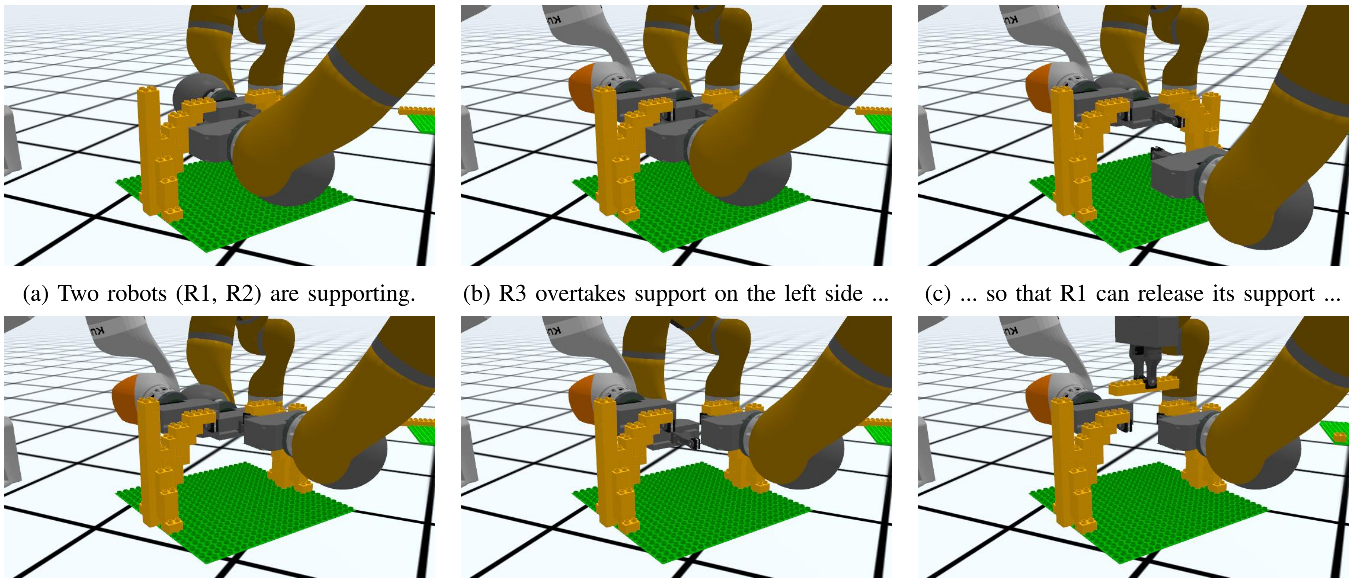
For enabling the planner to handle LEGO planning problems in general, a series of expert modules have been developed. *Analysis Modules* identify and extract LEGO-specific types of attributes from a given 3D model. The so-called *LegoPlacementAttribute* describes that a brick is being stacked on top of another with a given geometric offset. *LegoSupportAttribues* are used during planning and indicate that a robot is giving additional support to a brick in order to stabilize the construction. *Validation Modules* reject invalid planning states from further planning: While one LEGO-specific module aims at discovering exclusions as motivated in Fig. 1, another module also considers robots and performs statics analysis in order to determine the stability of a construction. Fragile constructions, which are not adequately supported by robots and, thus, would collapse due to their own weight, are reliably rejected.

The planning of micro steps uses *Skill Modules* to retrieve possible automation tasks that can be performed with robots in a current planning situation. For each robot, a *PickAndPlaceSkill* is provided that creates pick-and-place tasks in a given situation for each brick that is reachable by the robot. Such a task usually removes the *LegoPlacementAttribute* between a brick and its underlying bricks and constructs new ones at its new position respectively. In addition to the overall construction stability, all pick-and-place tasks furthermore require their underlying bricks to stand placement forces. As further skills for each robot, a *SupportSkill* and an *UnsupportSkill* provide tasks that construct or remove a *LegoSupportAttribute* between a robot and the construction. For providing bricks initially to the assembly process, suppliers have a skill named *SupplySkill* that constructs a *LegoPlacementAttribute* between a new brick and a supplier plate from which the brick can later be taken by a robot.

Equipped with these domain and automation modules, the planning approach is evaluated for each of the two LEGO problems. As the planning approach is implemented in Java, the evaluation is performed inside a JVM. The host machine provides 16 GB RAM and an Intel(R) Xeon(R) CPU E31230 (3.20GHz) 4-core processor with hyper-threading enabled. Each planning problem is evaluated in two different ways. First, evaluation runs are performed by a classical A* search strategy [17]. For the cost function in the LEGO domain, the overall execution times are used along with a kind of qualitative process reliability. Due to the immense variations of planning with multiple robots, the A* heuristics is an approximation that leads planning towards good results, not guaranteeing the very optimal solution. The A* search is equipped with all modules as introduced before to expand and validate the state space, enabling a comparison of its performance to our proposed MRPA that is used for a second evaluation. Here, evaluation runs profit from the corridor-guided search strategy and the early identification of deadlocks to avoid excessive planning time. For the planning with MRPA on micro step level, the same cost function and heuristic are used as for the A* evaluations. All evaluation runs longer than $300\,\mathrm{s}$ are discarded in both setups and are not considered in the computed evaluation statistics. To improve execution times of retrieved plans, a downstream optimization for multi-robot assembly is used to parallelise the strict sequential plan and allow asynchronous execution.

Table II shows the summarized evaluation statistics. For each evaluation, the number of successful runs and the average planning time with its standard deviation is recorded to analyse the planners' overall performance. For analysis of result quality, the number of domain and automation tasks respectively as well as the average execution time with the given number of robots – again complemented by its standard deviation – are shown. The LEGO house is planned by A* within about $2\,\mathrm{s}$ for one robot, resulting in a program with an execution time of almost $4\,\mathrm{min}$. Planned for three robots, A* actually finds shorter solutions where robots work in parallel. Planning times, however, increase non-linearly and find valid solutions in only 67 percent of all runs. Especially the standard deviation is even worse compared to the planning time. The deadlock problem of the roof corner stone is assumed to be responsible for that, as it might arise arbitrarily several times or not at all during planning.

In contrast to A*, MRPA provides results in all of its 250



(a) Two robots (R1, R2) are supporting. (b) R3 overtakes support on the left side ... (c) ... so that R1 can release its support ...

(d) ... and overtake the support of robot R2. (e) The now freed robot R2 can now ... (f) ... place the critical connection brick.

Fig. 6: Example of robot team cooperation as valid assembly solution found by the planner.

runs for both house evaluations – with one and with three robots. The planning time is drastically reduced, while finding the very same results as A* with one robot. With three robots, MRPA even finds far better solutions. The proportion of average planning time and its standard deviation is clearly improved with MRPA, expressing a reduced exposure of the planner to random good or bad decisions. The results of all evaluations with the house always count 52 domain tasks and 338 executed automation tasks, which indicates correct solutions as each of the 26 bricks requires a supply and a pick-and-place skill to be performed.

When planning the bridge, at least three robots are required in order to safely build the bridge structure with supports. A* returned a valid solution in only one of its 250 runs after approximately $4.5\,\mathrm{min}$, which seems to be a rather exceptional coincidence of lucky decisions that led to the solution. Although validation modules are used to reduce the state space, the bridge is still too complex for the A*-planner due to its huge set of possibilities, also owed to multiple robots and the supports needed in this example. In comparison, MRPA succeeds in all 250 runs within about a minute owing to its elaborate corridor-guided search strategy. In contrast to the house, variations in concretely chosen skills and tasks may apply, e. g., the specific supports that are actually performed. The average planning time, however, is still not exceeded by its standard deviation, which indicates rather reliable planning times. Although the high complexity of the planning problem, the overall planning time is remarkably good and still just a fraction compared to the execution time.

For building the bridge, a suitable set of robot cooperation is required to guarantee a non-collapsing assembly. A key moment of assembling the bridge is the placement of the uppermost brick of the arch which first connects both side parts. Here, the planner has to find a coordinated application of the right skills that never leave parts of the bridge unsupported (cf. Fig. 6). In the situation described by (a), two robots R1 and R2 are supporting each one part of the bridge at the second topmost brick. Robot R3 is not able to place the final connection brick as it would cause the directly underlying bricks to break. Thus, robot R3 moves in position to support one of these upper bricks (b), which enables R1 to release its support on this side (c). Repeating this scenario with Robot R1 on the other topmost brick (d), R2 is free to release its support (e) and to care about the critical connection brick which can now safely be placed (f).

VII. Related Work

In the field of planning, two different approaches have received great research interest. On the one hand, planning has been performed in state space, concentrating on the STRIPS formalism [18] and developments up to solvers for the planning domain definition language (PDDL) [19]. In these approaches, planning starts with a given initial situation, and searches for steps that lead towards a given goal. This principle allows performing planning as long as the possible steps can be defined, however the great amount of different step sequences often limits these approaches to rather small problems. To overcome them, various improvements have been suggested, from heuristics to guide search in a domain-specific manner [20] to abstractions that try to reduce the size of the planning system [21], however no solution for all of these problems has been found.

On the other hand, planning in plan space starts with a complex task description for the complete problem, and refines these tasks into combination of smaller tasks. Here, the concept of Hierarchical Task Networks (HTNs) [22] has found widespread use. These approaches make use of explicit abstractions given in the form of decomposition rules, which leads to more directed search and allows solving bigger problems, while requiring more domain knowledge incorporated into the planning process.

For use in realistic robotic applications, combinations of these approaches are often used, applying decomposition based planning for solving the overall symbolic task and assigning it to multiple robots, while search-based and sampling-based approaches are used to solve the continuous space motion planning [23], [24]. These solutions are applied to industrial robots that assemble parts [25], [26], [27], but also to mobile robots used in furniture assembly [28]. Still, they are only applied to relatively small problems (about ten parts). In contrast, the work of this paper combines these approaches with previous work of the authors, and focuses on relatively large planning problems that handle both symbolic planning on the domain level and geometric planning in continuous space.

VIII. Discussion and Future Work

In this paper, we presented a modular planning approach for coordinated multi-robot assembly. Besides a formal definition which allows for transfering the approach to different assembly domains, an overview of the underlying planning algorithm is given. We proposed several optimisation ideas to state-space planning that enable automated programming of coordinated multi-robot applications for assembly. The proposed concepts and the multi-robot planning algorithm have been successfully evaluated against planning problems in the field of LEGO. The overall planning performance as well as the resulting programs are convincing for even complex and challenging planning problems such as the introduced LEGO bridge, where three robots and cooperation is needed in order to safely build the structure.

Although collision detection is performed during planning for sequential programs, collision avoidance of asynchronously moving robots, e. g., when optimising the program, has not yet been addressed. Hence, only the house as been assembled in reality using one robot. All other assemblies using three robots are executed in simulation instead. Next steps will be the extension of collision detection also for optimised programs and, thus, the realisation of the assemblies with three robots. Moreover, we are working on transfering the planning approach to further domains, such as the assembly of airplane parts made of carbon-fibre reinforced polymers [4].

References

[1] *Manufacturing processes - Terms and definitions, division*, Deutsches Institut für Normung, 2020, DIN 8580:2020-01.

[2] N. Gupta and D. S. Nau, "On the complexity of blocks-world planning," *Artif. Intell.*, vol. 56, no. 2, pp. 223–254, 1992.

[3] A. Angerer, M. Vistein, A. Hoffmann, W. Reif, F. Krebs, and M. Schönheits, "Towards multi-functional robot-based automation systems," in *Intl. Conf. on Inform. in Control, Autom. & Robot.*, 2015.

[4] R. Glück, A. Hoffmann, L. Nägele, A. Schierl, W. Reif, and H. Voggenreiter, "Towards a tool-based methodology for developing software for dynamic robot teams," in *Intl. Conf. on Inform. in Control, Autom. & Robot.*, 2018.

[5] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif, "Robotics API: Object-oriented software development for industrial robots," *J. Softw. Eng. for Robotics*, vol. 4, no. 1, pp. 1–22, 2013.

[6] A. Hoffmann, A. Angerer, A. Schierl, M. Vistein, and W. Reif, "Service-oriented robotics manufacturing by reasoning about the scene graph of a robotics cell," in *45th Intl. Symp. on Robotics*, 2014.

[7] A. Hoffmann, A. Angerer, F. Ortmeier, M. Vistein, and W. Reif, "Hiding real-time: A new approach for the software development of industrial robots," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, 2009, pp. 2108–2113.

[8] L. Nägele, A. Schierl, A. Hoffmann, and W. Reif, "Automatic planning of manufacturing processes using spatial construction plan analysis and extensible heuristic search," in *Intl. Conf. on Inform. in Control, Autom. & Robot.*, 2018.

[9] ——, "Modular and domain-guided multi-robot planning for assembly processes," in *Intl. Conf. on Inform. in Control, Autom. & Robot.*, 2019.

[10] A. Björkelund, J. Malec, K. Nilsson, and P. Nugues, "Knowledge and skill representations for robotized production," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 8999 – 9004, 2011.

[11] M. Macho, L. Nägele, A. Hoffmann, A. Angerer, and W. Reif, "A flexible architecture for automatically generating robot applications based on expert knowledge," in *47th Intl. Symp. on Robotics*, 2016.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469 – 483, 2009.

[13] A. Skoglund, B. Iliev, B. Kadmiry, and R. Palm, "Programming by demonstration of pick-and-place tasks for industrial manipulators using task primitives," in *2007 Intl. Symp. on Computational Intelligence in Robotics and Automation*, 2007, pp. 368–373.

[14] F. Wildgrube, A. Perzylo, M. Rickert, and A. Knoll, "Semantic mates: Intuitive geometric constraints for efficient assembly specifications," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, 2019.

[15] L. Nägele, M. Macho, A. Angerer, A. Hoffmann, M. Vistein, M. Schönheits, and W. Reif, "A backward-oriented approach for offline programming of complex manufacturing tasks," in *6th Int. Conf. on Automation, Robotics & Applications*, New Zealand, 2015.

[16] L. Nägele, A. Schierl, A. Hoffmann, and W. Reif, "Multi-robot cooperation for assembly: Automated planning and optimization," in *Informatics in Control, Automation and Robotics, 16th International Conference, ICINCO 2019, Prague, Czech Republic, July 29-31, 2019 Revised Selected Papers*, ser. LNEE. Springer, 2020, to be published.

[17] S. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[18] R. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, no. 3/4, pp. 189–208, 1971.

[19] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL — the planning domain definition language," Yale Center f. Computat. Vision & Control, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. & Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[21] C. A. Knoblock, "Learning abstraction hierarchies for problem solving," in *Nat. Conf. on Artif. Intell.*, 1990, pp. 923–928.

[22] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "Shop2: An htn planning system," *J. Artif. Int. Res.*, vol. 20, no. 1, pp. 379–404, 2003.

[23] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," in *20^{th} Intl. Conf. on Automated Planning and Scheduling*. AAAI Press, 2010, pp. 254–257.

[24] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Intl. J. of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[25] U. Thomas and F. Wahl, "A system for automatic planning, evaluation and execution of assembly sequences for industrial robots," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Systems*, Oct 2001, pp. 1458–1464.

[26] B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang, "A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems," in *Intl. Conf. on Inform. in Control, Autom. & Robot.*, 2019.

[27] H. Fakhurldeen, F. Dailami, and A. G. Pipe, "Cara system architecture – a click and assemble robotic assembly system," in *Intl. Conf. on Robotics and Automation*, 2019, pp. 5830–5836.

[28] R. Knepper, T. Layton, J. Romanishin, and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *IEEE Intl. Conf. on Robotics and Automation*. IEEE, 2013, pp. 855–862.