DOI: 10.1109/AERO50100.2021.9438257
URL: https://ieeexplore.ieee.org/abstract/document/9438257

Please cite as:
I. Rodríguez, A. S. Bauer, K. Nottensteiner, D. Leidner, G. Grunwald and M. A. Roa, "Autonomous Robot Planning System for In-Space Assembly of Reconfigurable Structures," *2021 IEEE Aerospace Conference (50100)*, 2021, pp. 1-17, doi: 10.1109/AERO50100.2021.9438257.

Autonomous Robot Planning System for In-Space Assembly of Reconfigurable Structures

Ismael Rodríguez*
German Aerospace Center (DLR)
Institute of Robotics and Mechatronics
Münchner Str. 20, 82234 Weßling, Germany

Technische Universität München
Faculty of Informatics
Garching, Germany
ismael.rodriguezbrena@dlr.de

Adrian S. Bauer*, Korbinian Nottensteiner,
Daniel Leidner, Gerhard Grunwald, Máximo A. Roa
German Aerospace Center (DLR)
Institute of Robotics and Mechatronics
Münchner Str. 20, 82234 Weßling, Germany
firstname.lastname@dlr.de

***Abstract*— Large-scale space structures, such as telescopes or spacecrafts, require suitable in-situ assembly technologies in order to overcome the limitations on payload size and mass of current launch vehicles. In many application scenarios, manual assembly by astronauts is either highly cost-inefficient or not feasible at all due to orbital constraints. However, (semi-) autonomous robotic assembly systems may provide the means to construct larger structures in space in the near future. Modularity is a key concept for such structures, and also for reducing costs in novel spacecraft designs. The advantage of the modular approach lies in the capability to generate a high number of unique assets from a reduced number of building blocks. Thus, spacecrafts can be easily adapted to particular use cases, and could even be reconfigured during their lifetime using a robotic manipulation system. These ideas lie at the core of our current EU project MOSAR (MOdular Spacecraft Assembly and Reconfiguration).**

Teleoperating a space robotic system from Earth to assemble a modular structure is not straightforward. Major difficulties are related to time delays, communication losses, limited control modalities, and low immersion for the operator. Autonomous robotic operations are then preferred, and with this goal we propose a fully autonomous system for planning in-space assembly tasks. Our system is able to generate assembly and reconfiguration plans for modular structures in terms of high-level actions that can autonomously be executed by a robot. Through multiple simulation layers, the system automatically verifies the feasibility and correctness of action sequences created by the planner. The layers implement different levels of abstraction, hierarchically stacked to detect infeasible transitions and initiate replanning at an early stage. Levels of abstraction increase in complexity, ranging from a basic geometric description of the spacecraft, over kinematics of the robotic setup, to full representations of the actions. The system reuses information from failed checks in all layers to avoid similar situations during replanning. We use a hybrid approach where symbolic reasoning is combined with considerations of physical constraints to generate a holistic sequence of actions.

We demonstrate our planner for large space structures in a simulation environment. In particular, we consider the reconfiguration of a given modular structure, i.e. disassemble parts and reassemble them in a new configuration. The adaptability of our planning system is shown by executing the assembly plans on robots with different sets of skills and in scenarios with simulated hardware failures.

* Equal contribution to this work

978-1-7281-7436-5/21/$31.00 ©2021 IEEE

TABLE OF CONTENTS

1. INTRODUCTION

Space infrastructure has long sparked human imagination. Plenty of examples of large spacecrafts can be found in popular media, often being able to land on and lift-off from other planets. In reality, factors such as the cargo area of launch vehicles limit the size and mass of the payload, thus limiting the size of such spatial constructions. Moreover, structures optimized for use in space might not be able to endure gravitational forces on Earth, not even mentioning the high forces during launch. Thus, large space structures must be designed in modules and assembled in space, as it was the case for the International Space Station (ISS). Modularity and In-Space Assembly (ISA) can also be applied for instance for reconfiguration or maintenance of spacecrafts. However, in many application scenarios it would be highly cost-inefficient or even infeasible to have astronauts manually assembling modules or structures. Thus, (semi-) autonomous robotic assembly concepts are required.

Examples of projects aiming at developing such concepts are *MOSAR (MOdular Spacecraft Assembly and Reconfiguration)*[1] and *PULSAR (Prototype of an Ultra Large Structure Assembly Robot)*[2], both part of the European PERASPERA[3] program. PULSAR is motivated by the need of assembling larger space telescopes employing ISA techniques [1]. MOSAR, on the other hand, focuses on creating diverse spacecrafts by using reusable modules that can be reconfigured by a walking manipulator [2].

Teleoperating a robotic assembly system from Earth is not

[1] https://www.h2020-mosar.eu/
[2] https://www.h2020-pulsar.eu/
[3] https://www.h2020-peraspera.eu/

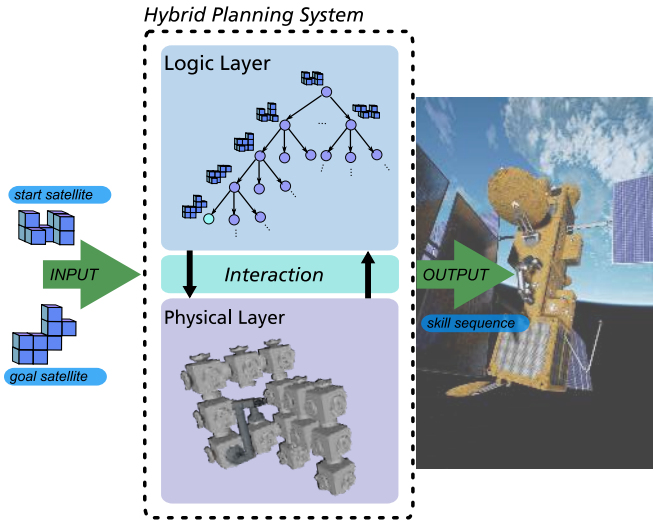


Figure 1: Visualization of the overall system. A hybrid planner is in charge of generating a robot skill sequence for reconfiguration, having as inputs the start and goal satellite configurations. Two main layers, a logical and a physical one, interact to fulfill this goal.

straightforward due to multiple reasons. First, time delay causes operators to adopt the *move-and-wait* strategy [3] starting with delays as small as 0.3s. While this would allow teleoperation of robotic systems in Lower-Earth Orbit (LEO), round-trip delays for communications with systems in geostationary orbit approach this value. Furthermore, low immersion and limited control modalities reduce the effectiveness of a human operator. Communication constraints such as packet losses, loss of signal, or jitter also hinder teleoperation.

Fully-autonomous systems are then a feasible alternative. In this paper, we present a fully autonomous system for planning in-space assembly tasks (see fig. 1). Based on the MOSAR scenario, the proposed system creates high-level action plans for reconfiguration of modular spacecrafts. The plans can be autonomously executed by a robot, as their feasibility is automatically verified through multiple simulation layers. The layers are organized hierarchically, with each layer representing different levels of abstraction. During planning, feasibility checks are executed sequentially on each level, from the highest to the lowest level of abstraction. This allows early detection of infeasible transitions, avoiding the waste of computational time on lower levels of abstraction, whose feasibility in general takes longer to compute. The levels in our approach range from a symbolic description of the environment, over robot kinematic considerations, to a full representation of the actions. The design of planning on multiple layers is tightly coupled to the concept of *hybrid planning* [4], [5].

We show that storing information about failed transitions accelerates planning and also that generalizing this information leads to great improvements in planning time. To achieve this, feedback from each level of abstraction is accumulated in *failure memories* to avoid similar situations during replanning stages. We further show that the system generalizes to different scenarios, e.g. different skill sets of the robot, different geometries, or failure of robot joints. Our system is evaluated on a scenario for reconfiguration of a modular satellite composed of cubical elements. Given the initial and target configurations of the satellite, the system computes a feasible plan to transform the satellite into the desired configuration. The general method is assessed by randomly generated problems, while hand-selected subsets of the problems are used to demonstrate the ability of the system to generalize to different skill sets.

The paper is structured as follows: First, work on related fields is introduced in section 2, followed by the required formal definitions in section 3. Then, we present the general planning system in section 4 and feedback strategies, used to learn from failures, are introduced in section 5. Lastly, the system is validated in experiments in section 6, and section 7 concludes the paper.

2. Related Work

This section provides an overview of the concept of modularity for space, and presents previous work on planners for assembly of modular satellites and hybrid planning methods.

Modularity in Space

Commercial space industry is moving towards reusability of systems and components [6]. Initial concepts for space modularity have been proposed, either through the development of re-usable subsystems or software modules [7], or through construction of different assemblies using commercial off-the-shelf components. This modular approach will enable changes in spacecraft construction and operation by providing more systematic approaches to On-Orbit Servicing (OOS) and On-Orbit Assembly (OOA). By performing maintenance, repair, and upgrade of components in space, the overall lifetime of satellites can be greatly extended, thus leading to a reduction of mission costs, increased reliability and sustainability, and quicker response of satellite developers to commercial demands.

The European Commission established the Strategic Research Cluster in Space Robotics Technologies, within the Horizon 2020 programme, to move forward in the development of advanced robotic space technologies. Three calls were foreseen within the cluster. The first call was focused on the development of common building blocks, which could be reused in the second and third calls, focused on applications in orbital and planetary scenarios. MOSAR is one of the projects of the second call, aiming to develop a demonstrator to test feasibility of modular and on-orbit reconfigurable satellites. The approach of MOSAR is based on re-use of hardware components and software generic building blocks, to perform mainly two demonstrations:

- Transfer of spacecraft modules from the cargo to the target satellite using a walking manipulator.
- Replacement of a damaged module on the target, with hot reconfiguration and re-routing of power and data.

Applications of robotic assembly in modular satellites are very recent, consequently little work has been published in this field and only few planners have been developed to specifically address this case. The iBoss project [8], [9] was a big step towards satellite modularization with the development of common building blocks and interfaces. iBoss also provided advances in the planning area, by introducing the PyHop[4] Hierarchical Task Network planning framework (Python implementation of SHOP [10]). This framework relies on a task-level planner that generates sequences of operations to solve the desired task. After the high-level

[4]https://bitbucket.org/dananau/pyhop

operations are planned, an inverse kinematic (IK) check and a motion planner are executed to compute the required robot trajectories. Different high-level strategies were provided in [11]. They generate reconfiguration transitions for satellites by classifying, melting, and sorting states. The robot is not included directly in the planning loop, but is considered by using a previous analysis of its capabilities in the high-level planning stage.

Hybrid planners

The planners for modular satellites discussed above do not intrinsically include the robot constraints in the system. In contrast, our planning system considers those constraints using a so-called hybrid planner, i.e. a Task And Motion Planning (TAMP) problem solver. These planners combine symbolic and geometric planning stages, as used for instance in the Asymov system [4]. Symbols are used to represent action preconditions in order to check whether an action can be performed by the robot. Often, geometric constraints are also represented with symbols in order to have preliminary feasibility checks at the symbolic level. For instance, [12] translates geometric poses into symbols for manipulation planning problems. The symbols are translated to be compliant with the Planning Domain Definition Language (PDDL, [13]), widely used by the planning community.

For TAMP problems, symbols can be used not only for expressing the geometric preconditions of a task, but also their effects. [14] uses a strategy where a search is performed for high-level tasks, and the output is tested with the motion planner. The same authors extended their work by presenting a system capable of learning the effects of parameterized skills, which allows the generation of skill sequences for solving more complex tasks [15]. The system can even perform preparation tasks that it foresees as necessary due to geometric constraints, for instance for pushing objects away to clear the path in order to retrieve a desired object. In addition, they improve the interrelation between the task and motion planner by using temporal logic constraints, which allows the representation of more complex scenarios.

The information of special-purpose reasoning can be integrated into symbolic planning using the "computed predicates" (also known as "semantic attachments") [16], [17], whose values are established by calling an external mechanism. This is aligned with our idea of checking the different movements of the robot with the motion planner and then storing whether it is feasible. Back-propagation of geometric failure situations to the symbolic layer using a hybrid planning algorithm was presented in [18]. A difference to our solution lies in how the back propagation (feedback) is exploited to solve future queries. We do not only detect and use these failures, but we store and re-use them to avoid the same problem in future re-planning actions.

Communication between the semantic or symbolic space and the geometric spaces (including physical capabilities of the real system) is required in the above-mentioned works. They mostly use pick-and-place problems as test cases, which have a reduced complexity compared to an assembly process involving several steps and tasks. On the other hand, in our previous works [19], [20], we intrinsically include the robots in the planning loop. We tested our approach in a more complex assembly domain, creating structures with modular aluminum profiles, thus proving the capabilities of our system to adapt not only to different assembly tasks, but also to different skill sets of the robot. For this work we take the idea of including the robot in the planning loop to reduce the amount of expert knowledge required to perform planning. This enhances the autonomy of the system, since it is then able to cope with a wider variety of problems and constraints with lower reliance on human knowledge.

3. FORMAL DEFINITIONS

This section provides the problem statement and the formal definition of the planning problem to solve. In particular, the state (i.e., configuration of a satellite) and the possible transitions are defined in order to model a sequence of reconfiguration steps executed with the help of robot skills.

Problem Statement

We consider as case study a modular reconfigurable satellite, as envisaged in the MOSAR project. Such a satellite consists of cube-shaped modules that carry different payloads, including for instance batteries, navigation systems, and experimental stations. We assume that each module is unique, and that the spatial arrangement of the modules can be reconfigured at any time so that it supports a specific application goal.

Figure 2: CEASAR robotic arm [21], whose base model is utilized in this work to design a walking manipulator. The end effectors are endowed with suitable standard interfaces to allow attaching and detaching actions.

In order to reconfigure a satellite, the system makes use of a robotic system, which in MOSAR corresponds to a walking manipulator. The robot considered here features seven degrees of freedom (fig. 2), and is capable of attaching and detaching to any of the six faces of a module through a standardized connection interface at each end. The robot can grasp and detach one module at a time, and place it at another location. The goal of the planner is to generate a sequence of steps to be followed by the robotic system to transform the initial configuration of a satellite into a desired one.

State Definition

Let us represent a module with index i as a tuple $m_i = (p_i, fixed_i)$, in which $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ is the position in the reference frame of the satellite, and $fixed_i = \text{const.} \in \{0, 1\}$ indicates if the module shall have a constant location or if it can be moved to another position by the manipulator. A satellite consists of M modules. A particular configuration of the satellite is given by a state $s_k = \{m_{i,k} = (p_i, fixed_i)_k \mid 0 \leq i \leq M\}$. A state is valid if each module is connected by at least one side to another module, and all

modules are interconnected.

Reconfiguration Sequence

Given the initial and goal configurations of the satellite, s_{start} and s_{end} respectively, the objective is to find a sequence of transitions t_k that move one module at a time, until the final configuration is reached. The configuration of the satellite is changed from s_k to s_{k+1} by transition t_k. The module to be moved needs to have at least one free face in s_k (and s_{k+1}) so that the manipulator can grasp it. Furthermore, only those transitions that do not violate the validity condition of a state are allowed. The walking manipulator receives the transitions as a set of tasks for execution. In order to solve these tasks, the robotic system provides a set of so-called skills, i.e. collections of atomic actions, which have the capability to generate trajectories considering the current state of the modules as geometric constraints.

Different robot embodiments have different sets of skills. In our case, three main skills are implemented: attach, detach, and move a module. By combining these three skills, higher level tasks can be performed. For instance, the robot can move itself on top of the satellite by attaching and detaching its end-effectors on the satellite connectors, a behavior we refer to as "walking" in the rest of this paper. Following the same logic, with a proper sequence of skills, tasks like moving a module from a starting to a goal position over multiple intermediate positions can be achieved.

4. AUTONOMOUS ROBOT PLANNING SYSTEM

In order to provide an assembly sequence plan, our planning system uses two abstraction layers, namely a logic and a physical layer. They are connected through a feedback system that enables the transfer of experience from the physical to the logic layer. An overview of the architecture of the planning system is first given, followed by a detailed description of the implementation of both layers.

Architecture

The architecture of our planning system is shown in fig. 3. It receives the start and goal configurations of the satellite as input. The planning system is tasked with finding a feasible solution for the reconfiguration sequence. The planner is composed of a logic layer, a physical layer, and a feedback system. The logic layer generates a state sequence, and it is in charge of all the semantic checks during the generation of this symbolic solution (list of states to reach the goal), i.e. it checks the validity of the states and required transitions. This layer does not have any information about the robot kinematics or any other geometric knowledge needed to plan and execute a particular transition with a robot motion. The physical layer takes a state sequence as input and performs checks related to the robotic capabilities and geometrical constraints. It includes IK checks and motion planning, embedded in planning levels of varying complexity. Both layers are presented in detail below.

A feedback loop is implemented to establish the communication between the two layers. In case of a failure in the physical layer, the feedback strategies (see section 5) are triggered. These strategies prune the search tree of the logic layer and also store the type of failure, in order to avoid it in future planning actions. This process runs iteratively until either a valid solution for reaching the desired state is found, or until all theoretically possible semantic solutions are checked. For practical reasons we set an upper limit on the number of allowed checks, since the number of feasible solutions grows exponentially with the number of modules.

Logic Layer

The logic layer is in charge of finding the semantic solutions considering different constraints. Such solutions can be seen as a sequence of states that change the system from the initial to the goal configuration. The core of this layer is based on graph search. The graph represents the possible states with the required transitions between them; therefore, a path through the graph represents a solution. Formally, the graph G is composed by nodes V and edges E. A node $v_i \in V$ represents a particular satellite configuration s_i, with v_{start} and v_{end} representing the start and end configurations, respectively. An edge $e_{ij} \in E$ connects the nodes v_i and v_j if and only if there is a valid semantic transition t_{ij} that connects the satellite configurations s_i and s_j. Finally, a semantic solution is a valid path, formally:

$$[v_{start}, ..., v_i, v_{i+1}, ..., v_{end}], \iff e_{i(i+1)} \in E \quad \forall i$$

In principle, multiple paths can be a valid semantic solution, but the shortest one is preferred as it requires the smallest number of transitions (therefore less robot motions are required). Based on this preference, the natural graph search algorithm to be used is the breadth-first search (BFS). In this algorithm, all the nodes that are one step further from the start are analyzed first, followed by the nodes that are two steps further, and so on. This method guarantees that the end node is reached with the minimal number of steps possible.

Using this basic version of BFS is enough to obtain a working system, but the huge amount of possible states and transitions between them makes the search slow for problems with a large number of modules. Two improvements are implemented to speed up this search. First, a heuristic is developed to give preference to states that are closer to the end state. Second, the branching factor (the average number of new states visited from a previous known state) is limited to b_{MAX} (fig. 4). Therefore, only the top b_{MAX} states are considered to search for a solution. The same heuristic is used to define the top states of the branching factor, and to choose which state should be visited next.

The heuristic for a state s_i is defined in the following priority order:

- The number of modules of s_i that are already in their final position.
- The number of modules of s_i that are not in their final position, but that are in a position that is free in s_{end}.
- The sum of the Euclidean distances of the module positions in s_i to their final position.

The idea of this heuristic is to quantify how far state s_i is from s_{end}. The first measure of how many modules are in a correct position is the simplest one. In case there is a tie in this measurement, we check how many modules are not occupying the final position of another module. This measure indicates how many modules must be moved to place another one in the correct place. Finally, in cases where the two previous measures do not differentiate two states, we simply check how far away the modules are from their final position. If a module is further away, it is more likely to require more transitions to take it to its final position, compared to a case where its final position is closer.

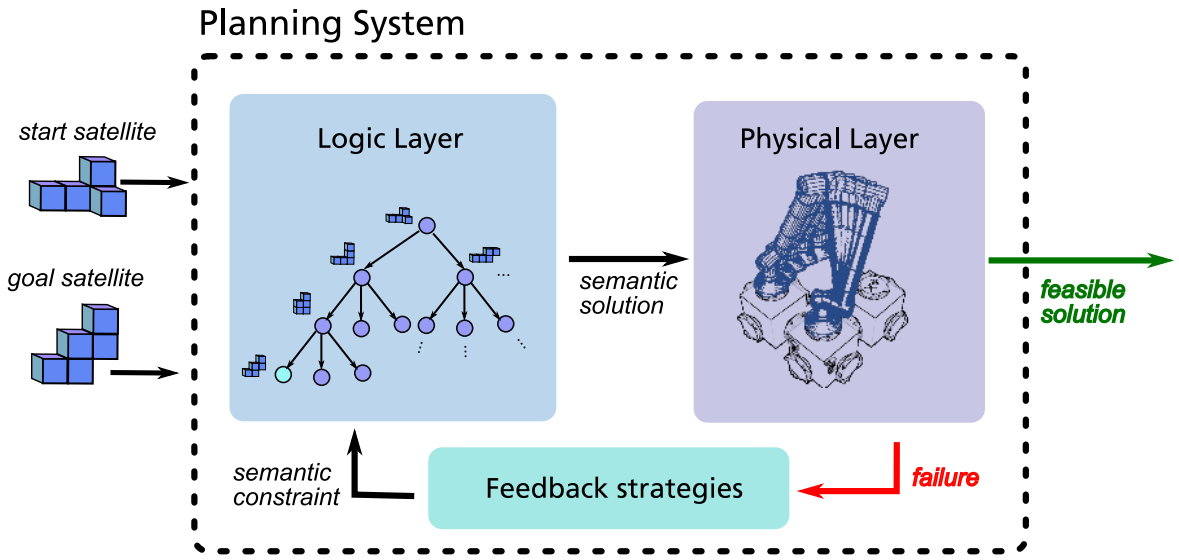


Figure 3: Architecture of the autonomous robot planning system. The system receives as inputs the start and goal satellite configurations, and iterates between the logic and physical layer until a solution is found.

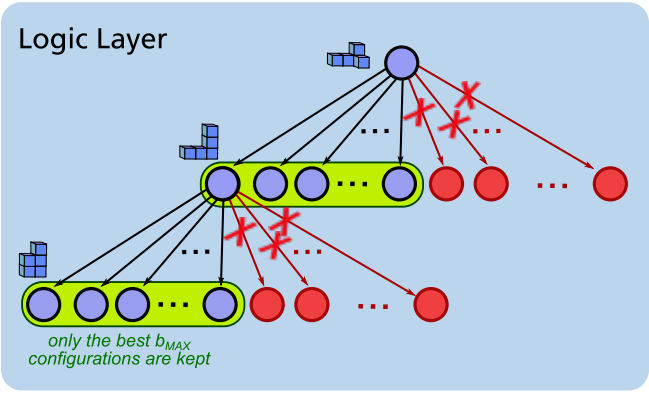


Figure 4: Concept of limiting the branching factor. In order to avoid the generation of too many new states, which negatively affect the overall planning time, a limit on the number of new states to keep is set. A heuristics is used to decide which are the fittest states.

Algorithm 1: Search for solutions in the logic layer.

```
1  PriorityQ_SAT ← ∅;
2  SEARCH LOGIC LAYER (s_start, s_goal, b_MAX)
3      s_actual ← s_start;
4      while s_actual != s_end do
5          PriorityQ_NEW ← GENNEWSTATES(s_actual);
6          for b_MAX do
7              s_new ← PriorityQ_NEW.pop(0);
8              s_new.parent ← s_actual;
9              PriorityQ_SAT.add(s_new);
10         end
11         s_actual ← PriorityQ_SAT.pop(0);
12     end
```

Algorithm 1 summarizes the search for solutions in the logic layer. The algorithm starts with an empty priority queue $PriorityQ_{SAT}$ (the priority used is the heuristic given before), which stores different discovered satellite configurations. The variable s_{actual} stores the configuration that is being analyzed at the moment, starting by the initial configuration s_{start}. In case the configuration under analysis is not the goal one (s_{goal}), an expansion is performed, checking for all the configurations that can be reached from s_{actual} by moving one module. These configurations are computed by the function GENNEWSTATES(s_{actual}). At the moment of generating the new possible satellites, the different constraints mentioned above plus the ones discovered in the physical layer are considered. The possible configurations are organized in a priority queue, and then the first b_{MAX} ones are stored in the main queue to keep the search until the goal configuration is reached. Finally, the path can be reconstructed reversely from the goal state by recursively following the parent.

Physical Layer

The logic layer abstracts away the underlying geometry, kinematics, and dynamics of the problem in order to plan efficiently. Thus, the physical layer's task is to ensure that solutions found on the logic layer are valid and feasible in the real world. For example, in [19], sub-levels of the physical layer are introduced to represent the planning problem with various degrees of complexity. There are levels that include collision checks, inverse kinematics, and dynamics of the task, each with only the parts, then adding the tools, and finally including the robot.

In the domain of cube-shaped modules with equal sizes, the objects can only be assembled in a predefined raster. Thus, static collision checks between the modules (without considering the robot motion) can easily be formalized and moved to the logic layer. By defining conditions on the construction of new edges in the graph, e.g. not allowing the robot to place modules at positions that are already occupied or only allowing to grasp modules from sides where there is no neighboring module, the logic layer filters out transitions that would result in a static collision.

Architecturally, the physical layer consists of two modules, one to check the inverse kinematics of the transition and one

Table 1: KEY POSES OF THE SKILLS

SKILL	KEY POSES
attach	• pose of target face at target module
detach	• no key poses because it does not include motion
move module	• pose of target face at module to move • pose of target face at goal position

to check if a feasible motion exists. The physical layer could be simply replaced by a motion planner, as the motion planner internally checks for IK solutions, i.e. if no IK solution to a given position exists, the motion planner does not try to find a valid trajectory to that point. However, inverse kinematics can be computed much faster than proving the existence of a valid motion plan, thus, we split the physical layer up into these two modules.

In order to evaluate the feasibility of a skill based on inverse kinematics, the IK module requires a definition of key poses to check. Thus, for each skill s we define a set of i_s key poses $k_i\,(p) \in SE(3)$

$$K_s(p) = \bigcup_{i=1}^{i_s} k_i\,(p)$$

where p is the skill parameterization. The key poses for our skills are listed in table 1. If any of the given poses of a skill cannot be reached, the physical layer returns a failure of type "inverse kinematics".

Motion planning is performed by a planner based on [22], which computes a valid trajectory between the start and end configuration of the robot that avoids collisions with the environment. The different skills of the robot are represented as motions between the defined key poses, with additional "attach" and "detach" actions that bind a module to or release a module from an end-effector. If the motion planner does not find a feasible trajectory for a given parameterized skill, it returns a failure of type "motion plan".

5. FEEDBACK STRATEGIES

The separation of planning into two layers brings various advantages. Computations are performed faster on the logic layer due to the use of symbolic characteristics. However, symbolic representations do not enclose all the information the system needs for a real execution. The physical layer plays the opposite role, by performing checks of higher fidelity, but being more expensive time-wise. In this work, we exploit the best of both worlds with what we call *feedback strategies*.

The feedback strategies collect information from the physical world and make them available on the logic layer. In this section, we present three different feedback strategies as exemplary cases. First, previously encountered IK queries are stored in a regression model for reachability prediction. The prediction model is based on the classical support vector machine (SVM) algorithm. Second, a failure memory is presented, which stores different cases where the motion planner was not able to compute a feasible trajectory. The structure of the satellite is analyzed and then compared with new cases to make a fail prediction of a new case without the need of using the motion planner again. Third, a pruning strategy for the search is given. Based on failures of previous semantic solutions, new solutions can directly be discarded.

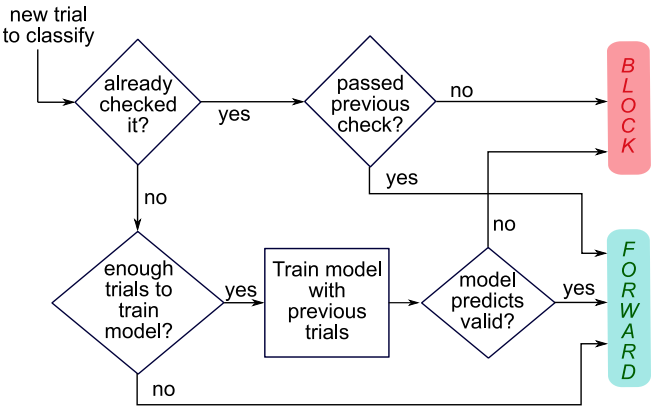


Figure 5: Decision flow for inverse kinematic regression. A new query is issued and, based on previous experience, a decision is made whether the query is forwarded to the IK solver or blocked.

A key aspect in the presented planning approach is its ability to learn from errors and avoid repeating them. In particular, queries to lower levels of abstraction are very expensive since they take longer to answer, thus, it is necessary to avoid them if they are known to fail. Thus, we implement *filters* that store failed queries and guard the gates of the IK module and the motion planner to keep them from expensively answering the same questions over and over again. This is inspired by the concept of two reasoning systems proposed in [23], a first one fast and intuitive, and a second one slower but more thorough. Our filters resemble system 1, while the motion planner and IK module resemble system 2. Importantly, we change this concept by making sure that our filters are overly conservative. This means that we rather accept false negatives than false positives. The reason for this is, that we prefer missing a possible solution over performing unnecessary checks. This is a trade-off between completeness and efficiency. As there are usually many solutions for the same problem, we expect the system to find one even though some valid solutions might be filtered out.

Inverse Kinematic Regression

One of the main goals of the different feedback strategies is to reduce the number of time-costly checks on the physical layer. Typically, IK checks are the most common checks performed on this layer. They can be understood as feasibility checks, i.e. whether the robot is able to reach a given Cartesian pose. These feasibility checks are performed at first for each movement that the robot has to perform. This verifies that the key poses of a movement can be reached, before spending time on searching for a valid trajectory – that might not exist – for that motion.

As the scenario presented in this work has a modular structure, it can be expected that similar IK checks are performed several times. In order to avoid solutions that have been proven to be infeasible before, it is required to translate this information into the logic layer. The modular structure of the problem allows for a simple look-up table that stores semantically infeasible solutions. However, a look-up table does not generalize to unseen cases, but it is only able to avoid doing exactly the same mistake again. To tackle the problem of analyzing new cases, the architecture of fig. 5 is proposed.

The only input of this system is the final pose of the end-effector of the robot, which is independent of the modules configuration. As mentioned above, if a particular position

has already been queried before, the system repeats the answer that was previously obtained. This method relies on the first experience obtained for a given position, but given the non-random nature of the IK solver and the modular nature of the problem, we do not expect any variation in the results. If a new (previously unknown) position is requested, the system has to make a prediction. It is expected for example, that if several positions close to the requested one are infeasible, the requested one is infeasible as well. Therefore, we implement a binary predictor.

Given the non-linearity of the problem, no simple hard-margin classifier can be used. Taking into account that our planner must be able to handle physical failures, the workspace of the robot under such restrictions can change drastically. For this reason, a binary prediction regression model with a SVM was chosen. The input for the algorithm is a vector $v = (x, y, z) \in \mathbb{R}^3$ that represents the target translation of the free end-effector with respect to the base of the manipulator attached to the satellite. Note that for the same vector v there can be multiple cases that classify differently because of the target orientation. The success probability of any vector v is defined as the amount of successful tests divided by the total amount of tests for that vector. The relative orientation could also be included into the input of the classification algorithm, but in order to improve data efficiency, the amount of features is reduced. As discussed above, we decided for a conservative approach. Thus, we define a threshold of success probability that any vector has to reach at least in order to be classified as feasible. In our system we empirically set that threshold to 0.1. In order to use the model it must first be trained. Thus, it rests as passive observer for the first trails and only later classifies new data based on the observed data. In our particular case we set this parameter to 20 trials. This threshold is a trade-off between the soundness of the classifier and, again, speed. In case that the model was not yet trained with enough cases, new queries are forwarded to the IK checker by default. The answer is then added to the training data.

We use this classification algorithm to restrict the search of solutions on the logic layer. Since the transitions between states already give an insight of the relative positions required, a fast check with the IK regression can be performed. The system looks over all the intermediate positions to find out if there is one where it is feasible to reach the initial and goal positions of the transition. An example is shown in fig. 6, where a module has to be moved from position A to position B. In the first try there is an IK failure that indicates that the system cannot reach the final position. By changing the intermediate position, the robot can now reach the starting and goal configurations. If no intermediate position can be found where the movement is feasible, then the transition is directly discarded in the logic layer. If the IK regression does not classify the position as feasible, it can be skipped. Thus, the transitions that pass the classifier very likely do not have IK issues.

Another advantage of this approach is its re-usability. If the parameters of two tasks are the same (i. e. robot kinematics and environment), the information collected before is still valid. Therefore, if a target pose is requested, the memory can be used without modifications.

Failure Memory

As discussed above, it is interesting to keep lower modules from answering the same queries over and over again. However, we design the failure memory to remember failed

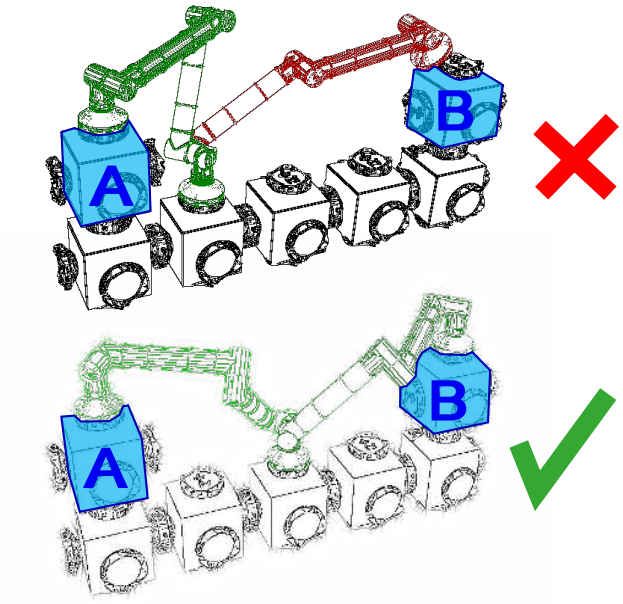


Figure 6: Inverse kinematic checks performed to verify if it is possible to translate a module from position A to position B. In the first case (above) this is not possible, but it is solved by changing the intermediate position in the second case (bottom).

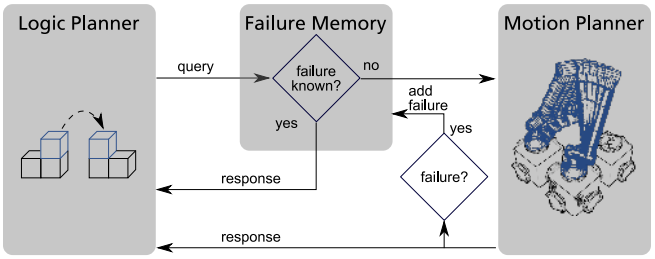


Figure 7: Concept of the failure memory. The logic planner queries the failure memory with a transition. If the failure memory already knows that this transition is impossible, it immediately returns this. Otherwise it forwards the query to the motion planner. The motion planner replies to the logic module and updates the failure memory if a failure is found.

feasibility checks; not to store feasibility checks that succeed. This is due to safety concerns. Our system is designed to work in space applications, thus the generated plans must guarantee success. Therefore, every transition in the final plan must be validated by all lower level modules, and we do not want to rely on – partially probabilistic – methods of the failure memory. On the other hand, if the failure memory is wrong in classifying a transition as infeasible while it actually is feasible, the worst thing that can happen is that the planner misses a solution. The idea of this concept is visualized in fig. 7.

The most common types of actions the robot performs in our scenario are attaching to and detaching from modules. As detaching does not include any motion of the robot but solely changes the state of the connector at the end-effector, it can never fail and is, thus, not considered further in the realm of the failure memory. Attaching the robot to a module means that the robot is already attached to a module with one end-

effector and moves its other end-effector to a specified face of another (sometimes even the same) module. The failure memory represents such a transition of type "attachment" as a tuple $\mathcal{T} = \langle \mathcal{W}, g_i, g_f \rangle$, where $\mathcal{W}$ is the world state, g_i is the index of the target module, and g_f is the target face. Furthermore, we define the world state as $\mathcal{W} = \langle S, r_0, r_1, f_0, f_1 \rangle$ where $S = \{c_1, \dots c_n\}$ is the satellite state containing the indices of the individual modules c_i, r_j is the index of the module the robot is attached to with end-effector $j \in \{0, 1\}$, and f_j is the face to which end-effector j is attached. The modules are attached to each other in a fixed raster, which allows to view their positions as discrete variables. The position of module i is thus $p_i \in \mathbb{Z}^3$, the same holds for r_0 and r_1. We did not include the robot configuration into the world state for reasons that will be discussed later.

We identified two crucial requirements to ensure that the failure memory has the desired effect: (i) the failure memory has to detect queries that have already led to negative answers, and (ii) it must be able to generalize from situations that lead to negative answers in new but similar situations.

The first requirement translates into the need of having an identity function that behaves like the Kronecker delta for world states:

$$\delta_{\mathcal{W}_1, \mathcal{W}_2} = \begin{cases} 1 & \text{if } \mathcal{W}_1 = \mathcal{W}_2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

If such a method exists, the planner can save failed transitions along with the state in which they occurred, and prevents from testing them again. In the given problem the requirement of an identity method is fulfilled because of the discrete world state according to our definition, which makes it easy to compare two world states. In continuous domains each variable of the world state additionally requires a threshold that represents the amount of deviation that is accepted between two world states while still being considered identical. In hybrid planning approaches, discretization is typically achieved by means of symbolic state representations [4], [24]. In our system, eq. (1) in combination with a unique representation of the transition allows to identify queries that have already been answered.

Restricting the failure memory to only recognize identical situations is obviously not sufficient. Thus, we designed the failure memory to generalize to new but similar situations. The approach we use to achieve this goal is to "remove" ambiguity in the world state and transition: First, ambiguity due to the freely defined world origin is removed by aligning the world frame with the robot coordinate system. Thus, the world is centered on the robot and the orientation of the robot in the world is fixed using the initial state of the robot. Second, we remove ambiguity introduced by objects out of reach of the robot. Therefore, all modules of the satellite that are further away from the robot origin than the maximum length of the robot are removed from the world state. Since the robot cannot collide with these modules, they do not restrict its possible motions and thus, they can safely be ignored.

Third, we exploit symmetry of the robot to further reduce ambiguity. Most robotic systems are designed with some symmetry, thus, what we describe in detail for the given robot can be transferred to other robots as well. In our specific case we make use of two symmetries: First, we assume the robot is symmetric with respect to the fourth joint, so it does not make a difference which end-effector is connected to a module and which one is used in the attach action. Second, the first and the last axis of the robot can be rotated from $-\pi$ to π. Thus, robot attachment positions can always be chosen such that the next goal is in a specified quadrant if projected onto the x-y plane (in robot coordinate frame). This approach is conservative and can be over restrictive, but as discussed above, this is the result of a trade-off between soundness, efficiency, and completeness of the plan.

The world state considered for the failure memory has then the following characteristics: (i) the satellite is centered around the module, (ii) the satellite does not contain modules out of reach of the robot, and (iii) the satellite is rotated such that the goal is in positive x and y direction. Figure 8 visualizes the approach used for generalization.

The generalized states can later be used to check whether a transition is feasible in a specific satellite configuration. Consider that the failure memory is aware of a failure for a transition given a generalized satellite configuration s_a, and the question is whether it also fails given the current satellite configuration s_b. This can be answered by using the generalized representation of s_b. If then $s_a \subseteq s_b$, it can be concluded that the transition also fails in s_b. All the modules in s_a, which possibly played a role to prevent the skill from being executed successfully, are also present in s_b, thus the transition will fail.

The approach of the failure memory can be generalized to other actions and problems if an identity function for world states is designed. Furthermore, the efficiency of the failure memory can be improved by means of removing unneeded information in the world state.

As mentioned above, the failure memory does not take the robot joint configuration into consideration. This is because joint information is only required if there are two or more distinct regions in the configuration space of the robot where transitions from one region to the other are not feasible. We argue that this does not happen often in our scenario and, again, underline the conservative behavior of the failure memory. As our experiments show, this conservatism helps to reduce planning time enormously.

Pruning Search

Another way to make use of feedback is to prune solutions that have been shown to fail. Pruning is key in problems where the amount of semantically valid solutions grows exponentially with each step. Also, since it is expensive timewise to check solutions on the physical layer, it is important to eliminate all cases known to fail in the layer above (see fig. 9).

The key point for a proper pruning is how to choose what solution should be pruned and where, based on the infeasible transition. In our system we used the simplest approach, all states that are under the infeasible transition t_{inf} are discarded. In a formal way, all the solutions that have the same transitions as the failed solutions up to the transition t_{inf} are pruned. Since all the steps up to that transition have been the same, the same infeasibility is expected to happen.

A difference between this feedback strategy and the two previously mentioned is its lack of reusability. It is less likely to have a similar transition between the same states as in the failed plan given the huge size of the configuration space for the satellite.

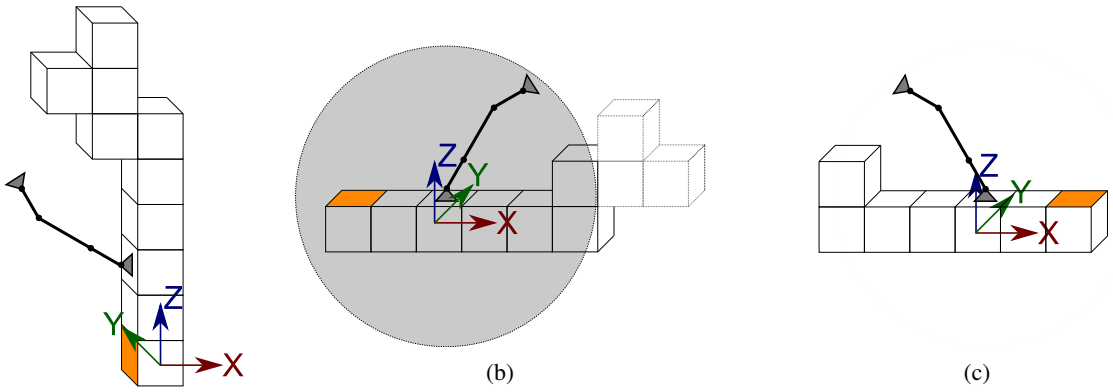


Figure 8: Steps for generalizing a transition. (a) shows the original configuration. The robot's task is to attach to the orange face. In (b) the world is centered on the robot, and modules out of reach are removed. Next, in (c) the satellite is rotated in order to have the target position in positive x and y coordinates.

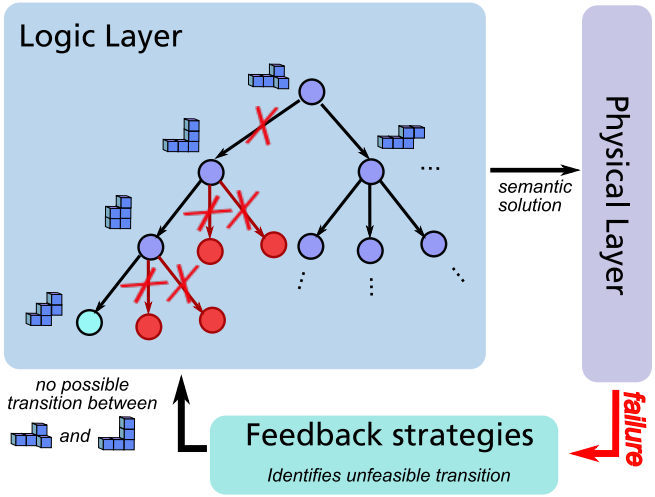


Figure 9: Concept of pruning. Once a transition was found to be infeasible in the physical layer, this information is sent back to the logical layer for the corresponding pruning. All the states that were built using such transition are discarded.

These three strategies have been shown as examples of how the two layers can interact, exploiting the best of both worlds. We believe that the fact of separating the task generation in a logic layer from the execution of robot in a physical layer allows the system to adapt to different interactions between robot and satellite. In the following section, different experiments show such adaptation and how the feedback strategies reduce the number of checks performed in the physical layer, therefore reducing the planning times.

6. Experiments and Validation

To verify the performance of our system, we ran multiple experiments. The two key points we demonstrate are the reduction of checks required when the feedback strategies are used, and the capability of the system to adapt to different scenarios. We show the first point by comparing the planning time of our system with and without feedback. The second point is evaluated by designing and running three different scenarios that require adaptation. The three scenarios we consider are (i) changing the size of the cubic modules, (ii) reducing the skill set of the robot, and (iii) simulating a joint failure.

In order to achieve a fair comparison, we created random tasks for rearranging satellites consisting of 4 to 15 cubic modules. For each number of modules in the satellite, we created 5 test scenarios. While investigations on the feedback strategies are reported over all test cases, we pick exemplary test cases to show the adaptation of the system. All experiments ran on a 64-bit Linux computer with Intel® Xeon® CPU E5-1630 v4 @ 3.70GHz with 8 cores and 16 GB RAM. Our system was not optimized to achieve the best planning times; the focus was rather on demonstrating the usefulness of feedback, and learning from feedback during planning.

Time Reduction using Feedback Strategies

The impact of the feedback strategies described in section 5 is evaluated by comparing planning times with and without them. The focus of this work does not lie on comparing effects of different feedback strategies, but on demonstrating their usefulness. Thus, we compare two scenarios: running the system without any feedback strategies versus running the system with IK regression and the failure memory. Since the feedback strategies do not affect planning time on the logic layer but on the physical layer, and in our test cases semantic search showed to be much faster than physical simulation, we only compare planning times on the physical layer.

In space applications that do not require online calculations it is unusual to compare approaches based on their absolute execution times, as super computers on Earth can be employed to perform the required computation. However, the search space for symbolic solutions grows exponentially with the size of the problem, therefore even super computers reach their limits when trying to solve a sufficiently complex problem. Consequently, the more effective the planning algorithm is, the more complex problems can be solved.

We performed the evaluation on 12 different levels of difficulty, ranging from 4 to 15 modules per satellite. As described above, we created 5 random test problems that the system had to solve for each level of difficulty. The results are reported in table 2. Note that the absolute times are not comparable, as there is a lot of variation. Some test cases seem to be easy and fast to solve, while others are harder to solve and, thus, take longer. This can be seen by the

high standard deviation reported for the physical planning times. Importantly, we group the tests by the number of modules in the satellite, while the difference between initial and final state (i.e. the amount of modules to move) is chosen randomly. Therefore, we focus on relative time savings in the following. Note also that the time spent on the logic layer can be neglected in comparison to the time spent on the physical layer, thus we focus on the results for the physical layer.

Typically, the feedback strategies do not affect the time required for the logic layer. For tests with low numbers of modules we see a slight increase in planning times (e.g. with 5, 6, 7, 9, and 10 modules) that can be explained with the overhead added through the feedback strategies. In contrast, for tests with a higher number of modules (e.g. with 8, 11, 12, 13, 14, and 15 modules) the time spent on the physical layer decreases slightly. This comes as the effect of pruning unfeasible solutions at an early stage thanks to feedback strategies outweighs their overhead. All in all the effects of the feedback strategies on the time spent on the logic layer can be neglected with regards to their effects on time spent on the physical layer.

The results show that implementing feedback can save tremendous amounts of time during planning. In average, the feedback strategies saved 27.5% of planning time, with a standard deviation of 13.0%. The numbers also show that the effect of the failure memory and the IK regression vary strongly. In the test cases with 5 modules, for example, the failure memory did not reduce the amount of queries to the motion planner at all. In contrast, for the case of 14 modules (which seems to be a hard problem to solve based on the planning time of the logical layer), the failure memory blocked approx 22.2% of the calls. The reason for these extreme differences could be that, on the one hand, the problems with 5 modules were very easy to solve (as also the absolute times suggest) and there were no infeasible transitions. The problems with 14 modules, on the other hand, were much more difficult and contained many similar transitions that were filtered out by the failure memory.

The IK regression was rather constant in the ratio of calls blocked. While this ratio was low for relatively simple cases with 4, 5, 6, or 8 modules, it showed its effectiveness for more complex scenarios, with ratio of blocked calls ranging from 23.91% in the case of 7 modules to 33.35% in the case of 14 modules. Note also that the relative time savings grow with the difficulty of the problem, as shown in fig. 10. This suggests that the feedback strategies are especially useful for complex problems. Overall, this experiment shows the importance of feedback strategies in planning.

Adaptation to Different Scenarios

To illustrate the ability of our system to adapt to different scenarios, we consider three types of changes that require system adaptation, and demonstrate them in experiments. The scenarios presented in this section might also serve future designers of modular satellites or robotic systems in their design choices.

Changing size of modules—One decision that designers of modular satellites have to take is the geometry of the modules. Therefore, it is key to test how the robotic setup interacts with different modules and if it is able to perform the desired tasks. Naturally, it is desired that the system can adapt to different geometries, making it more autonomous, rather than manually adapting the planning system for each geometrical variation. Geometric adaptation also enables the system to deal with new modules if designers decide to change module geometry at some point.

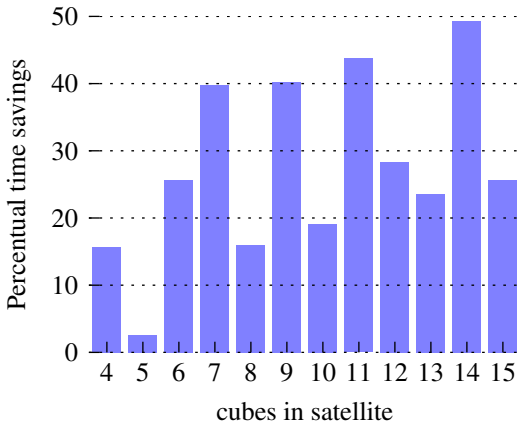


Figure 10: Time savings (in percentage of total motion planner time) when using feedback strategies

In our experiment we changed the edge length of the modules from 0.6 m to 0.8 m. The change of module size suffices to enforce different solutions for many problems. We tested the performance of the system with the new module geometry in multiple of the test cases defined for the first experiment. In some of the cases, the resulting plans where identical to the original plans because the original plan was still feasible. However, to illustrate the adaptability we present a test case with 5 modules that yielded different solutions. The results with original and larger module size can be compared in table 3. While the plan is the same on the highest level of abstraction, i.e. the move actions are the same, it differs on the next level, which contains walking actions. Since the distances to cover are bigger in the case of the bigger modules, the robot needs an extra walking motion to reach the goal position. This demonstrates that the system is able to adapt to different geometries and is still able to find solutions.

Different skill sets—The proposed system is also able to adapt to different skill sets. The skill set of a robot can change over lifetime, as programmers implement new skills or old skills are removed due to degradation of a robot. Also, a designer might have the choice between different robots with different skill sets to perform a task. Our planner can answer whether a certain robotic system can handle a given situation with its skills or not. In particular, we demonstrate this in a case were the walking manipulator is no longer able to walk, but is fixed to the structure with one end. We tested this scenario on multiple test cases from the original experiment. The planner did not find a solution for any problem with more than 6 modules. This is because in those tests some modules are too far away for the robot to reach when its base is fixed. We use a test case with 6 modules to show, exemplary, how the different solutions look like. The solutions can be compared in table 4.

The most obvious difference between the plans lies in the missing walking actions with the reduced skill set, which was expected as the walking skill was the skill that we deactivated in the experiment. Furthermore, it can be observed that the planner decided for another intermediate position to store module 3. While it seems that the heuristics used during planning encouraged the planner to walk towards the target module before moving it in many instances of the original

Table 2: EVALUATION OF FEEDBACK STRATEGIES FOR DIFFERENT SCENARIOS.

Difficulty # of modules	WITHOUT FEEDBACK avg. time logic layer in s	WITHOUT FEEDBACK avg. time physical layer in s	WITH FEEDBACK avg. time logic layer in s	WITH FEEDBACK avg. time physical layer in s	WITH FEEDBACK calls blocked by failure memory in %	WITH FEEDBACK calls blocked by IK regression in %	time savings in %
4	0.09	65.65 (±55.4)	0.09	55.34 (±46.2)	8.84	15.97	15.70
5	0.21	46.89 (±20.6)	0.23	45.65 (±19.4)	0.00	8.04	2.65
6	7.73	56.73 (±22.9)	8.03	42.18 (±14.5)	1.04	8.72	25.66
7	7.17	339.83 (±157.4)	7.39	204.79 (±49.9)	16.01	23.91	39.74
8	1.94	130.54 (±72.4)	1.90	109.64 (±40.1)	2.67	15.87	16.02
9	1.32	380.62 (±189.7)	1.40	227.69 (±46.1)	3.88	27.27	40.18
10	12.46	441.01 (±284.4)	12.88	356.73 (±175.0)	20.08	25.63	19.11
11	22.95	773.43 (±501.7)	22.22	435.06 (±162.6)	11.82	29.45	43.75
12	10.31	704.04 (±286.1)	10.13	504.74 (±89.1)	12.97	32.65	28.31
13	14.78	1689.89 (±849.6)	14.63	1291.03 (±908.0)	21.35	33.13	23.60
14	280.95	1904.94 (±820.4)	269.78	965.08 (±171.2)	22.22	33.35	49.34
15	26.78	659.35 (±96.2)	26.56	489.86 (±91.9)	3.41	26.64	25.71

Table 3: GEOMETRICAL ADAPTABILITY OF THE PLANNER: TEST CASE WITH TWO DIFFERENT MODULE SIZES (SEE APPENDIX FIG. 11 AND FIG. 12).

ORIGINAL CUBE SIZE	BIG CUBES
walk from pos [0,0,0] using face x-	walk from pos [0,0,0] using face x-
walk via pos [0,0,0] using face y+	walk via pos [0,0,0] using face y+
	walk via pos [1,0,0] using face y+
walk to pos [1,0,1] using face x+	walk to pos [1,0,1] using face x+
MOVE CUBE 3 from [2, 0, 0] to [0, 0, 2] using face: x+	MOVE CUBE 3 from [2, 0, 0] to [0, 0, 2] using face: x+
MOVE CUBE 2 from [1, 0, 0] to [0, 1, 0] using face: x+	MOVE CUBE 2 from [1, 0, 0] to [0, 1, 0] using face: x+

Table 4: ADAPTABILITY TO DIFFERENT SKILL SETS: COMPARING ONE ORIGINAL CASE AND ONE WITH A REDUCED SKILL SET (SEE APPENDIX FIG. 14 AND FIG. 15).

ORIGINAL SKILL SET	REDUCED SKILL SET
walk from pos [0,0,0] using face x-	
walk via pos [0,0,0] using face y-	
walk to pos [1,0,0] using face x+	
MOVE CUBE 5 from [0, 1, 1] to [0, 2, 0] using face: x+	MOVE CUBE 5 from [0, 1, 1] to [0, 2, 0] using face: x-
MOVE CUBE 4 from [1, 1, 0] to [0, 1, 1] using face: x+	MOVE CUBE 4 from [1, 1, 0] to [0, 1, 1] using face: z+
MOVE CUBE 3 from [0, 0, 1] to [1, 1, 0] using face: x+	MOVE CUBE 3 from [0, 0, 1] to [1, 1, 1] using face: y-
walk to pos [0,2,0] using face x+	
MOVE CUBE 2 from [1, 0, 0] to [0, 0, 1] using face: x+	MOVE CUBE 2 from [1, 0, 0] to [0, 0, 1] using face: y-
MOVE CUBE 3 from [1, 1, 0] to [1, 0, 0] using face: z+	MOVE CUBE 3 from [1, 1, 1] to [1, 0, 0] using face: y-

setup, the planner was also able to find a solution without walking. This shows that our system is able to adapt to different skill sets.

Joint failure—Another scenario that any system deployed in space or in orbit must be able to deal with are failures. In order to have an operative system for as long as possible, it is important that the system can still be operated if some failures occur. In particular we analyze the case of a joint failure, where one joint can only be moved in a limited range.

For the experiments we reduced the joint limits of the fourth joint of our robot to a range from $40°$ to $180°$ instead of originally $-180°$ to $180°$. With the modified limits, the minimal step size of the robot during walking increases and the robot is no longer able to make small steps to neighboring module faces. As with the other experiments, we tested this scenario on a subset of the original test cases. With the reduced joint limits the robot was still able to finish all tests successfully, but the solutions differed from the original ones.

An exemplary test case with 5 modules is shown in table 5. In the solution of the original problem, the robot is attached to face x+ of the module at [1, 0, 1] before moving module 3. Directly afterwards, the robot tries to move the neighboring module at [1, 0, 0] with face x+, which is possible in the original setup. With the reduced joint limits, however, the robot is not able to attach to this face and has to walk around first, until it is attached to face y+ of module [1, 0, 1], and from this location it can perform the move action. Thus, this illustrates how the system is also able to cope with robot failures.

7. CONCLUSIONS AND FUTURE WORK

This paper presented a hybrid planning system with a logical and a physical layer, capable of generating plans to transform a modular satellite from an initial to a final configuration. The logic layer is in charge of the high-level planning of the tasks,

Table 5: ADAPTABILITY TO JOINT FAILURES: COMPARING ONE ORIGINAL CASE AND ONE WITH A REDUCED RANGE OF MOTION FOR ONE JOINT (SEE APPENDIX FIG. 11 AND FIG. 13).

NORMAL JOINT LIMITS	REDUCED JOINT LIMITS
walk from pos [0,0,0] using face x-	walk from pos [0,0,0] using face x-
walk via pos [0,0,0] using face y+	walk via pos [0,0,0] using face y+
walk to pos [1,0,1] using face x+	walk to pos [1,0,1] using face x+
MOVE CUBE 3 from [2, 0, 0] to [0, 0, 2] using face: x+	MOVE CUBE 3 from [2, 0, 0] to [0, 0, 2] using face: y+
	walk to pos [1,0,1] using face y+
MOVE CUBE 2 from [1, 0, 0] to [0, 1, 0] using face: x+	MOVE CUBE 2 from [1, 0, 0] to [0, 1, 0] using face: x+

while the physical layer deals with the robot capabilities and geometry problems. The two layers are interlaced through a feedback loop, which improves the planning time by quickly discarding infeasible solutions.

We have demonstrated the ability of our system to adapt to different scenarios without any core changes. Three main cases were shown: changing the size of the modules of the satellite, simulating a joint failure, and using a different skill set for the robot. However, the adaptation capability comes at the cost of requiring several time-consuming checks in the physical layer. Different strategies were implemented to reduce the amount of such checks. Experiments have shown that these strategies have reduced planning times in average by 27.5% and in some cases up to 50%.

Although the feedback strategies have shown a great impact on the planning times, there is still room for improvement. Additional feedback strategies can be implemented to retrieve more information from the physical layer in order to reduce the planning time even more. Another possible improvement is to reduce the communication between different subprograms, which, in our case, was a major source of time consumption due to our middleware. Note also that our current implementation was not optimized to give the best time performance; parallel programming could significantly improve this aspect. In any case, the goal of this work was not to set a baseline for planning problems in terms of absolute times, but to demonstrate the usefulness of integrating feedback from the physical layer on the logic layer.

We believe that this work can be the basis for a simulation and testing environment for new robotic systems designed to work with modular satellites. Comparisons between different robotic setups can be drawn on how they behave with new module sizes, on how they react to failures, how many skills need to be implemented, and so on. Our architecture supports such studies by simply adapting the modular planning system, instead of having to implement a completely new one.

ACKNOWLEDGMENT

The authors would like to gratefully acknowledge the financial support and endorsement from the DLR Management Board Young Research Group Leader Program and the Executive Board Member for Space Research and Technology.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 821966, project MOSAR.

REFERENCES

[1] M. A. Roa, K. Nottensteiner, G. Grunwald, P. L. Negro, A. Cuffolo, S. Andiappane, M. Rognant, A. Verhaeghe, and V. Bissonette, "In-space robotic assembly of large telescopes," in *Proc. 15th Symp. Advanced Space Technologies Robotics and Automation (ASTRA)*, 2019.

[2] P. Letier, X. Yan, M. Deremetz, A. Bianco, G. Grunwald, M. Roa, R. Krenn, M. Munoz, P. Dissaux, J. Garcia, R. Ruiz, L. Filippis, G. Porceluzzi, M. Post, M. Walshe, and P. Perryman, "MOSAR: MOdular Spacecraft Assembly and Reconfiguration demonstrator," in *Proc. 15th Symp. Advanced Space Technologies Robotics and Automation (ASTRA)*, 2019.

[3] W. R. Ferrell, "Remote manipulation with transmission delay," *IEEE Trans. Human Factors in Electronics*, no. 1, pp. 24–32, 1965.

[4] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *Int. J. Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[5] J. A. Wolfe, B. Marthi, and S. J. Russell, "Combined Task and Motion Planning for Mobile Manipulation." in *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*, 2010, pp. 254–258.

[6] J. P. Davis, J. P. Mayberry, and J. P. Penn, "On-orbit servicing: Inspection, repair, refuel, upgrade, and assembly of satellites in space," *The Aerospace Corporation Center for Space Policy and Strategy*, 2019.

[7] J. Ocón, F. Colmenero, J. Estremera, K. Buckley, M. Alonso, E. Heredia, J. García, A. Coles, A. Coles, M. Martínez, E. Savaş, F. Pommerening, T. Keller, S. Karachalios, M. Woods, S. Bensalem, P. Dissaux, A. Schach, R. Marc, and P. Weclewski, "The ERGO framework and its use in planetary/orbital scenarios," in *Proc. Int. Astronautical Congress (IAC)*, 2018.

[8] M. Kortman, S. Ruhl, J. Weise, J. Kreisel, T. Schervan, H. Schmidt, and A. Dafnis, "Building block based iBoss approach: fully modular systems with standard interface to enhance future satellites," in *Proc. Int. Astronautical Congress (IAC)*, 2015.

[9] M. Goeller, J. Oberländer, K. Uhl, A. Roennau, and R. Dillman, "Modular robots for on-orbit satellite servicing," in *Proc. 13th Symp. Advanced Space Technologies Robotics and Automation (ASTRA)*, 2016.

[10] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proc. Int. Conf. Artificial Intelligence*, 1999, pp. 968–973.

[11] Y. Zhang, W. Wang, J. Sun, H. Chang, and P. Huang, "A self-reconfiguration planning strategy for cellular satellites," *IEEE Access*, vol. 7, pp. 4516–4528, 2018.

[12] R. Dearden and C. Burbridge, "Manipulation planning using learned symbolic state abstractions," *Robotics and Autonomous Systems*, vol. 62, no. 3, pp. 355–365, 2014.

[13] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL-the planning domain definition language," *AIPS98 Planning Competition Committee*, 1998.

[14] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2014, pp. 3684–3691.

[15] L. P. Kaelbling and T. Lozano-Pérez, "Learning composable models of parameterized skills," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2017, pp. 886–893.

[16] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *Towards service robots for everyday environments*. Springer, 2012.

[17] C. Dornhege, A. Hertle, and B. Nebel, "Lazy evaluation and subsumption caching for search-based integrated task and motion planning," in *IROS Workshop AI-based robotics*, 2013.

[18] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014, pp. 639–646.

[19] I. Rodríguez, K. Nottensteiner, D. Leidner, M. Kaßecker, F. Stulp, and A. Albu-Schäffer, "Iteratively refined feasibility checks in robotic assembly sequence planning," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1416–1423, 2019.

[20] I. Rodríguez, K. Nottensteiner, D. Leidner, M. Durner, F. Stulp, and A. Albu-Schäffer, "Pattern recognition for knowledge transfer in robotic assembly sequence planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3666–3673, 2020.

[21] A. Beyer, G. Grunwald, M. Heumos, M. Schedl, R. Bayer, W. Bertleff, B. Brunner, R. Burger, J. Butterfaß, R. Gruber, *et al.*, "CAESAR: Space robotics technology for assembly, maintenance, and repair," in *Proc. Int. Astronautical Congress (IAC)*, 2018.

[22] P. Lehner and A. Albu-Schäffer, "The Repetition Roadmap for Repetitive Constrained Motion Planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3884–3891, 2018.

[23] D. Kahneman, *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.

[24] D. Leidner, C. Borst, and G. Hirzinger, "Things are made for what they are: Solving manipulation tasks by using functional object classes," in *Proc IEEE-RAS Int. Conf. Humanoid Robots*, 2012, pp. 429–435.

BIOGRAPHY

Ismael Rodríguez *received his degree of Ingeniero en Electrónica from Universidad ORT Uruguay in 2015. Since 2016 he is pursuing a Ph.D. in Robotics at the German Aerospace Center (DLR) in Weßling. His research is focused on the development of an assembly planner that is aligned with the requirements of the mass customization phenomenon.*

Adrian S. Bauer *received his B.Eng. degree in Mechanical Engineering in 2012 from DHBW Ravensburg, a B.Sc. degree in Cognitive Sciences in 2015 from the University of Tübingen, and a M.Sc. in Robotics, Cognition, Intelligence from the Technical University Munich in 2018. Since 2018 he is pursuing a Ph.D. in Robotics at the German Aerospace Center (DLR) in Weßling.*

Korbinian Nottensteiner *holds a Dipl.-Ing. (Univ) degree in mechatronics and is currently pursuing his Ph.D. at TU Munich. He leads the team for autonomous assembly systems of the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR). His primary research goal is to enable autonomous robotic assembly by situation-aware manipulation skills using compliant control, contact sensing and task abstractions.*

Daniel Leidner *received his Ph.D. degree in artificial intelligence in 2017 from the University of Bremen, Germany. His dissertation was honored with the Georges Giralt PhD Award as well as the Helmholtz Doctoral Prize. He is currently leading the Fault-Tolerant Autonomy Architectures Group at the Institute of Robotics and Mechatronics of the German Aerospace Center (DLR), where he investigates artificial intelligence in the context of astronaut-robot collaboration.*

Gerhard Grunwald *received his Diploma and Doctoral Degree in Computer Science from the Karlsruher Institute of Technology (KIT, formerly University of Karlsruhe). Since 1988 he has been with the Institute of Robotics and Mechatronics at the German Aerospace Centre (DLR). Dr. Grunwald has coordinated several national and international cooperative research projects focusing on robotics for industrial, professional, and domestic services. Since 2012 he is heading the Orbital Robotics activities of the institute. This comprises robotized active debris removal, assembly, maintenance, and repair.*

Máximo A. Roa-Garzón *received his doctoral degree in 2009 from Universitat Politecnica di Catalunya, and the Project Management Professional (PMP) Certification in 2016. He worked for Hewlett Packard R&D before joining the Institute of Robotics and Mechatronics of the German Aerospace Center (DLR) in 2010, where he leads the group on Robotic Planning and Manipulation.*

APPENDIX

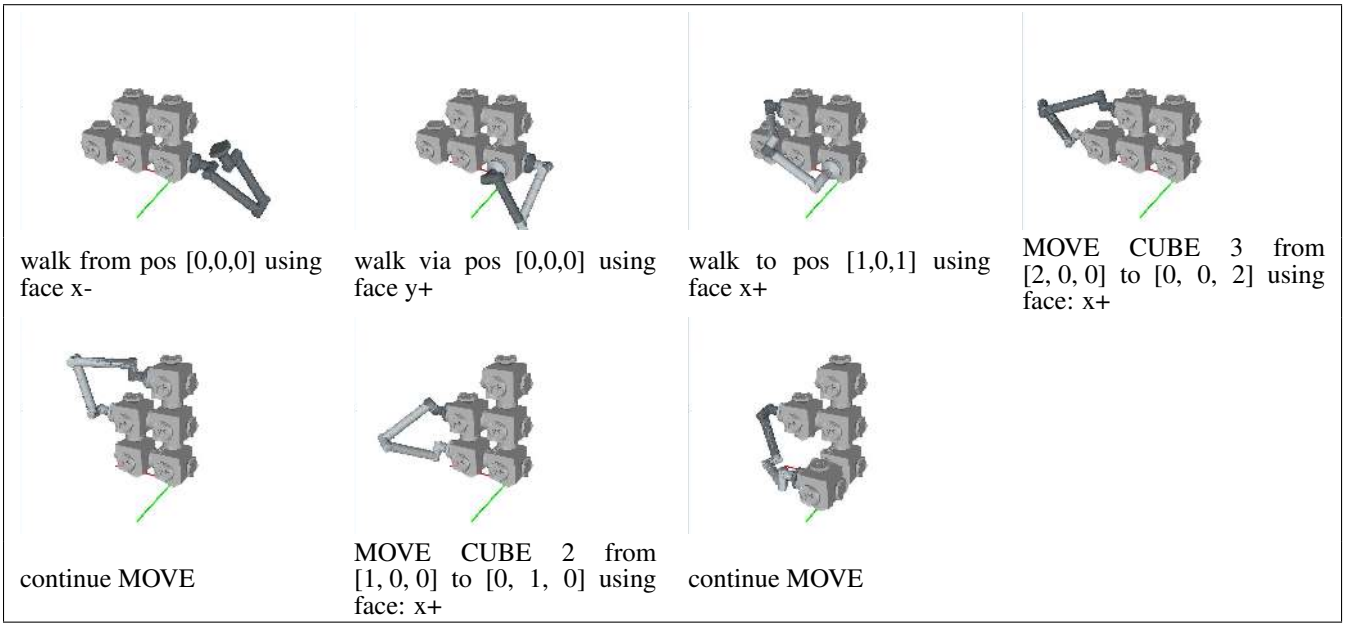


Figure 11: Reference solution for problem with 5 modules.

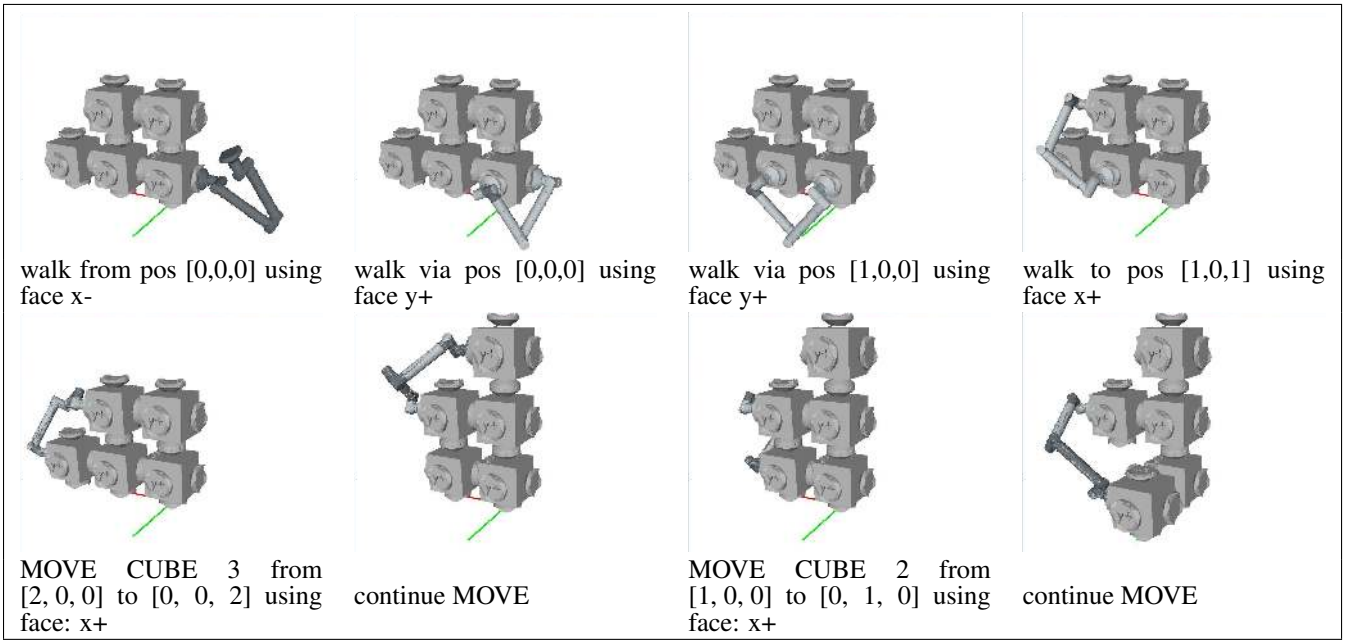


Figure 12: Solution for problem with 5 modules and increased module size.

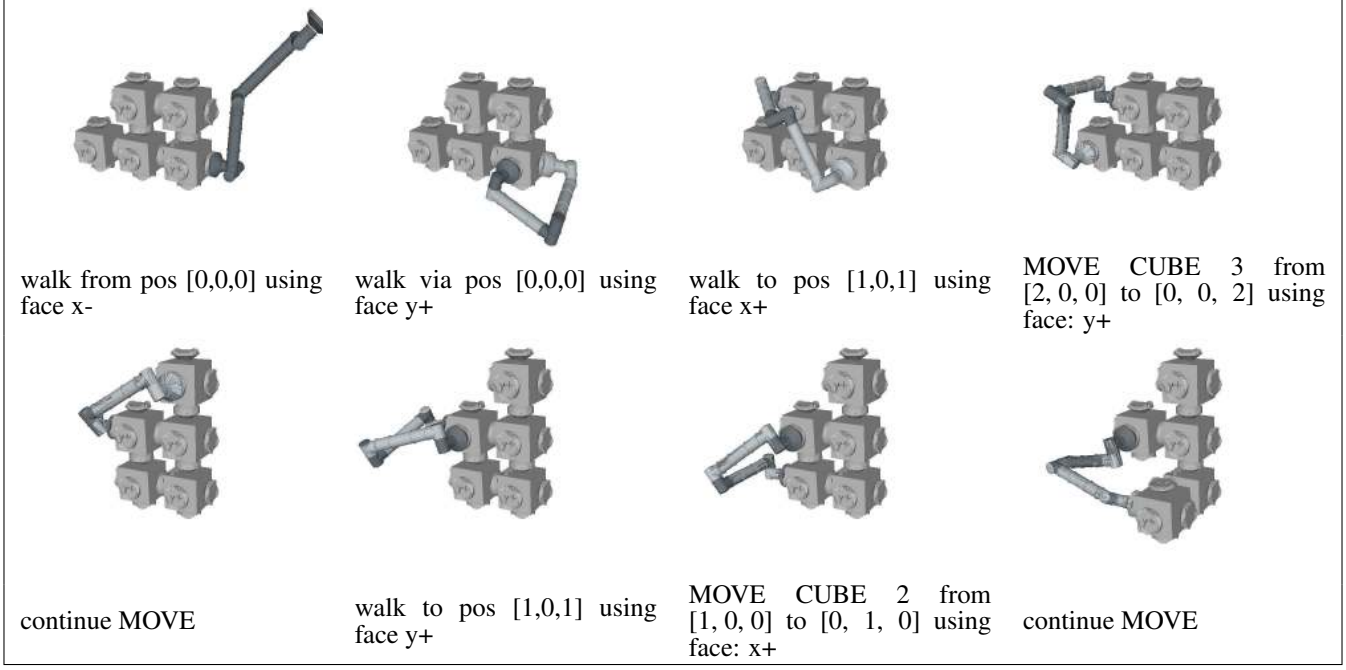


Figure 13: Solution for problem with 5 modules and reduced joint limits.

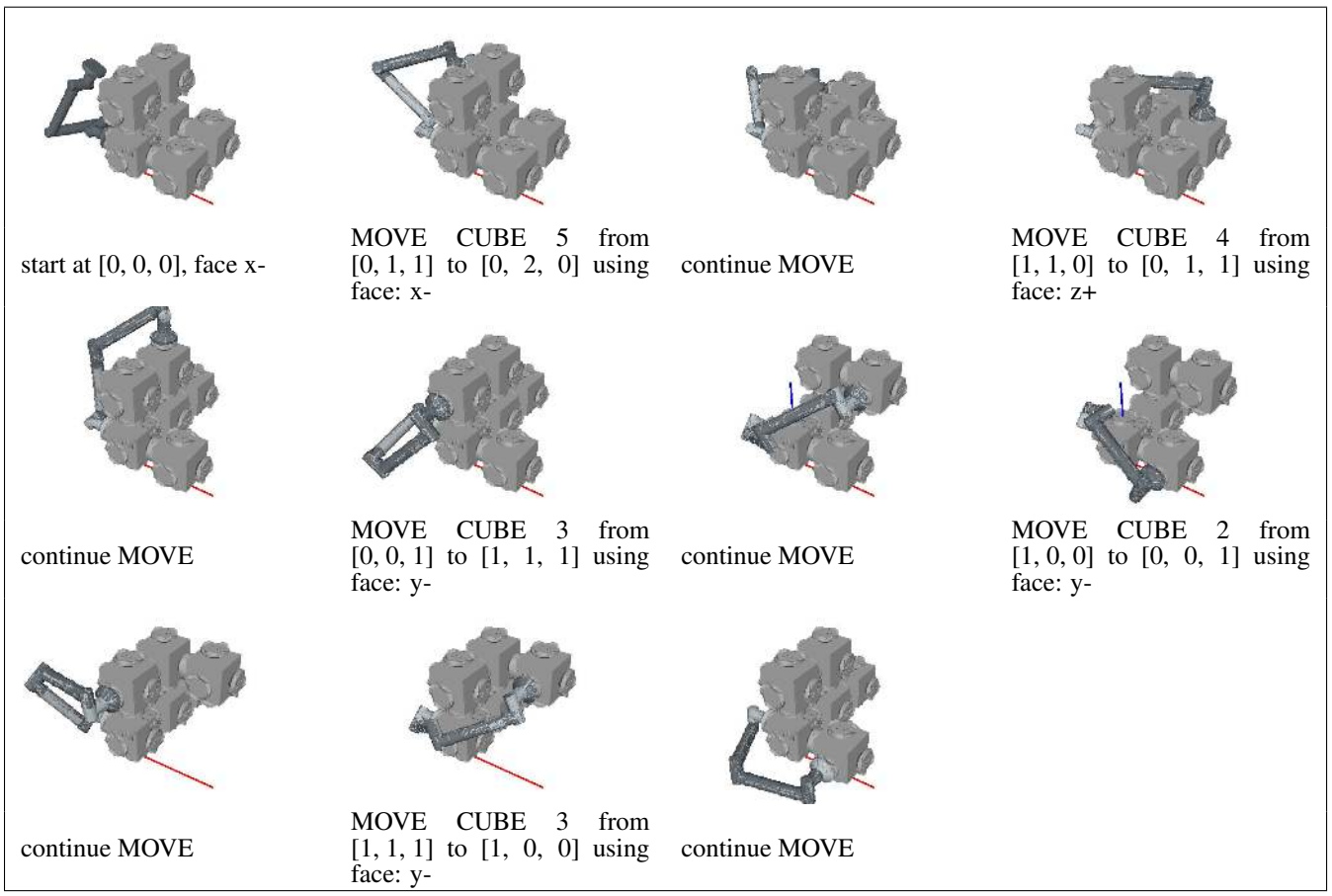


Figure 14: Solution for problem with 6 modules and reduced skill set.

walk from pos [0,0,0] using face x-

walk via pos [0,0,0] using face y-

walk to pos [1,0,0] using face x+

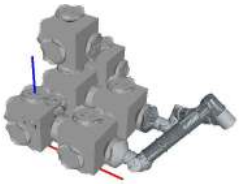
MOVE CUBE 5 from [0, 1, 1] to [0, 2, 0] using face: x+

continue MOVE

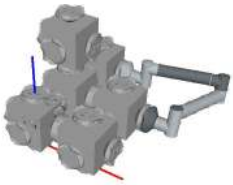
MOVE CUBE 4 from [1, 1, 0] to [0, 1, 1] using face: x+

continue MOVE

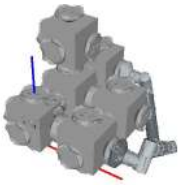
MOVE CUBE 3 from [0, 0, 1] to [1, 1, 0] using face: x+



continue MOVE

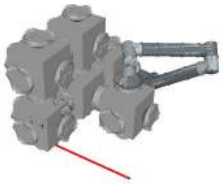


walk to pos [0,2,0] using face x+



MOVE CUBE 2 from [1, 0, 0] to [0, 0, 1] using face: x+

continue MOVE



MOVE CUBE 3 from [1, 1, 0] to [1, 0, 0] using face: z+

continue MOVE

Figure 15: Reference solution for problem with 6 modules.