Learning and
Intelligent
Systems

Technische
Universität
Berlin

Combined Task and Motion Planning for Modular Self-Reconfiguring Robots

Cornelius V. Braun

Abschlussarbeit
zur Erlangung des akademischen Grades

Bachelor of Science
Informatik

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Learning & Intelligent Systems Lab

Erster Gutachter: Prof. Dr. Marc Toussaint
Zweiter Gutachter: Prof. Dr. Oliver Brock
Betreuer: Dr. Ozgur S. Oguz

8. August 2021

Abstract The vast majority of the robotic systems that are used in the real world cannot be reconfigured to execute novel tasks. Thus, in factories or households, often a different robot is used for each task. However, the purchase of one robotic device for each task comes with high cost for the users and high amounts of electronic waste are produced. Modular robots find a remedy to this issue. This type of robot can be assembled from standardised and cheap units, their shape can be reconfigured to optimally fulfil a given task, and modular robots can be easily repaired by simply exchanging the broken modules. So far however, modular robots are not ready for many real-world scenarios because planning actions and motion execution are usually performed separately, which limits the capacities of those systems dramatically. The present work introduces a universal and integrated long-horizon task and motion planner for modular robots. Using the planner, the robot modules can autonomously decide when and how to reconfigure, and manipulate the environment to achieve a high-level goal. In order to effectively find a plan for task execution, we leverage geometric information from the environment that guides symbolic planning. To further increase long-term planning capacities of our planner, we present an iterative version that uses a receding horizon. Our experiments show that the modular robotic system is able to complete various tasks faster than non-modular robots. By introducing integrated task and motion planning for modular robots, we aim to increase the possible fields of application of modular robots.

Zusammenfassung Die meisten Roboter, die heutzutage eingesetzt werden, können nicht an neue Aufgaben angepasst werden. Daher wird in der Industrie oder auch in Haushalten oft für jede Aufgabe ein eigener Roboter verwendet. Die Anschaffung eines Roboters für jede einzelne Aufgabe ist jedoch mit hohen Kosten für die Nutzer verbunden, zudem fallen große Mengen an Abfall an. Modulare Roboter schaffen Abhilfe bei diesem Problem. Diese Art von Robotern kann aus standardisierten und billigen Einzelteilen zusammengebaut werden, ihre Form kann rekonfiguriert werden, um eine gegebene Aufgabe optimal zu erfüllen, und modulare Roboter können leicht repariert werden, indem die defekten Module einfach ausgetauscht werden. Bislang kommen modulare Roboter jedoch noch nur selten zum Einsatz, da Handlungs- und Bewegungsplanung in der Regel getrennt voneinander erfolgen, was die Fähigkeiten solcher Systeme erheblich einschränkt. Zur Lösung dieses Problems, wird in dieser Arbeit ein universeller und integrierter Langzeit Handlungs- und Bewegungsplaner für modulare Roboter vorgestellt. Mithilfe des Planers entscheiden die Robotermodule autonom, wann und wie sie ihre Struktur ändern und die Umgebung manipulieren, um eine Aufgabe zu erfüllen. Um die Handlungsplanung effizienter zu gestalten, nutzen wir geometrische Informationen über die Umgebung. Um die Langzeitplanungsfähigkeiten unseres Planers zu verbessern, stellen wir eine iterative Version vor, die einen zurückweichenden Horizont verwendet. Unsere Experimente zeigen, dass das präsentierte modulare Robotersystem mithilfe des Langzeitplaners in der Lage ist, verschiedene Aufgaben schneller zu erledigen als nicht-modulare Roboter. Durch die Einführung eines integrierten Handlungs- und Bewegungsplaners für modulare Roboter wollen wir die Anwendungsmöglichkeiten modularer Roboter erweitern.

Contents

1 Introduction

The advances in robotics research have lead to robotic devices now being integrated in workflows of various life domains and industries. Progress in hardware development has made the deployment of robotic systems for commercial and private use safe and increasingly popular [1], [2]. On construction sites, the usage of robots yields higher construction speeds and helps to improve worker security in dangerous industries [3]. Articulated robots are used for various tasks across industries like welding in the automobile manufacturing process [4] or pick-and-place operations in the packaging industry [5]. Medical robots have been employed in heart surgery [6] and rehabilitation of patients with neurological disorders [7]. In the examples above, robotic devices ease the life of their owners, increase the safety in the working environment and even save lives of many patients.

However, a major problem of most modern robotic systems is that pipelines involving robotic manipulators are typically predefined for a fixed set of tasks and require extensive set-up. Thus, when a production pipeline is altered, often a new device is needed or the entire application needs readjustment. Modular robots are a solution to these shortcomings. A modular robot is an individual robot, that is assembled from several smaller robotic units, which may or may not be able to act as independent robots on their own, if not connected [8]. Such systems can easily be altered to perform various tasks, e.g. the Thorvald II system, which can be configured as fruit picking robot or as field tillage robot [9]. Consequently, with modular robots there is no need for a completely new device when work processes change, and the system is more robust to failures, as a defective unit can easily be replaced by a new one [8]. Furthermore, the modifiable shape of modular robots can be exploited to increase the mobility of a system [10], [11] or to adapt to new environments [12], [13]. Furthermore, modular robots offer cost-reduction, as simple and standardised modules can be mass-produced and a small range of modules can be used to assemble many different robots [14]–[16].

Some modular robots are even able to reconfigure their own individual robotic units completely autonomously [11], [17]. These robots are called *modular self-reconfiguring robots* (MSR) [8], [18]. Illustrating the full potential of modular robots, Yim et al. [18] described a long-term vision for MSR called "bucket of stuff". In this vision, consumers have several cheap and run-of-the-mill, self-reconfigurable robot modules at home and can simply request the robots to "clean the gutters" or "change the oil in the car". The robots then plan all actions, motions and robot configurations that are necessary to complete the task on their own and finally perform it. However, to this day, the existing systems are still very far from being able to complete such as task. In fact, modular robotic manipulation has been restricted to very few and simple actions in standardised and simple environments so far [10], [11], [17], [19]–[22]. A main reason for this is the lack of *task and motion planning* algorithms for modular robots that allow to plan and execute such a task.

To enable robots performing more complex tasks in cluttered environments, a lot of research in the artificial intelligence and robotics community has been dedicated to improve *task planning* [23], [24] and *motion planning* [25] for non-modular robots. *Task planning* deals with finding a sequence of discrete actions that a robot should execute in order to achieve some predefined goal, e.g. finding all the acts required to successfully

brew a cup of coffee, such as pouring water into the boiler and grasping a cup. Once a plan is found, the robot needs to perform additional *motion planning* in order to be able to move from one state to another. The key to performing any task fully autonomously, like in the "bucket of stuff vision", lies in solving task and motion planning jointly, which is called *integrated task and motion planning* (TAMP). TAMP is crucial for just any type of autonomous robotic manipulation, ranging from simple tasks like picking up an object and placing it on a desk or complex tasks like assembling a car from scratch.

Whilst scenarios like pick-and-place tasks [26], stacking boxes [27], [28] or navigating through unstructured environments [29] can be planned and performed by non-modular robots in reasonable time, every day tasks such as loading and unloading a dishwasher or making coffee are still proving to be really difficult for any robot, and there is no TAMP algorithm that can solve such a problem in a suitable amount of time to this day [30]. Furthermore, despite of the exciting assets of modular robotic systems and the importance of TAMP for autonomous robotic systems in robotic applications, to our knowledge there is no integrated TAMP algorithm that has planned long-horizon tasks of modular robots so far. Therefore, with the present work, our main contribution is to provide a flexible and broadly applicable long-horizon integrated task and motion planner for modular robots. We demonstrate that the framework we use is able to handle heterogeneous links between robots with different topologies and even links of robots and non-robotic structures like struts. Furthermore, we show that our planner is able to plan long-horizon tasks for modular and non-modular robots alike.

The present work

The framework for TAMP that we use in the present study is called *logic-geometric programming* (LGP) [27]. This approach combines symbolic search-based planning methods with optimisation-based motion planning [1]. In order to end up with an effective LGP-based task and motion planner that handles modular robotic structures, we had to address several challenges.

First of all, the symbolic planner used in LGP, as previously implemented in [27], [31], is not capable of effective motion planning of modular robots due to the combinatorial complexity of those scenarios. The number of potential actions, robot configurations and states increases exponentially w.r.t. the number of agents in the environment, which was already an issue for TAMP of non-modular robots [32]. As modular robotic systems entail high degrees of freedom and a high number of actors, task and especially motion planning of these systems is very complex. A major contribution of the present work is to leverage geometric information from the environment that the robots are acting in, with the aim to prune the search tree during LGP planning. To this end, we introduce a problem independent interface for heuristics for LGP. The heuristic functions that we deploy with this interface use the initial positions of objects and robots in the environment to compute heuristic cost-to-go values that guide an informed search of the symbolic action space. Thereby, geometrically infeasible paths in the decision tree are not expanded, which can speed up the task planning component of LGP. Thus, we effectively introduce the usage of geometry-based heuristics in the LGP framework with the present work.

[1] A detailed description of this procedure will be provided in a later section.

The second major contribution of the present work is the demonstration that long-horizon TAMP for modular robots is possible and effective. The main challenge that comes with the deployment of a modular robotic system arises from the reconfiguration of the individual modules. When rearranging a modular robotic system, a new kinematic structure is created, which often comes with reprogramming the entire modular robot [33]. In contrast, the LGP-based approach that we present is able to handle not only self-reconfiguration, i.e. the autonomous reconfiguration of the robot modules, but also the kinematic constraints that come with this type of planning problem. To achieve this, we formulate optimisation constraints on the connection of two (or more) robots. These constraints ensure that the connection between the robot modules is stable and physically plausible. Upon connection, the kinematic chains of the modular robots are handled such that a virtual joint is created at the juncture and the two kinematic chains of the individual robots are unified. In addition, we present several symbolic actions for task planning, such as *connect* and *disconnect*, which can only be executed when multiple robotic units are in the same environment.

Third, we present a variation of our TAMP planner that plans tasks with a limited horizon approach. Big challenges for long-horizon TAMP are changes in the environment during planning [34], [35] and planning such long motion sequences [36], [37]. In order to enable our planner to take on these challenges, we iteratively solve the TAMP problem. Iterative TAMP has the benefit that the symbolic action space size is reduced [38]. In addition, executing only a limited amount of motions at each step increases the robustness of motion planning, as optimisation-based motion planning difficulty is with the number of planned actions [36]. We call this approach *Receding Horizon Logic Geometric Programming* (RH-LGP), as it is inspired by the Receding Horizon Control (RHC) method, which is typically used to continuously control process by solving optimisation problems a sliding time window. At each step, RHC only predicts a plan of action for a fixed of future states. By sliding this time window into the future, there new outputs are predicted to continuously control the process [39]. Similarly, we define a limited horizon h for motion planning, such that at each iteration only the first h actions of a symbolically feasible plan are actually executed. Then, the optimisation window is moved and the next h actions are planned based on the kinematic outcome of all previous actions. By introducing this variant of the planner, we demonstrate that it can solve long-horizon TAMP problems for modular and non-modular robots alike.

The remainder of this work is organised as follows: First, we review related work on modular robotics, TAMP in robotics, heuristics in TAMP and long-horizon TAMP. Then, we describe the details of our planner, followed by experiments that demonstrate the efficacy of the framework. Finally, we discuss the limits of our contributions and consider some ideas for future work to extend the presented planning method.

2 Related work

Relevant works related to the present study can be grouped into four categories: modular robotics, combined task and motion planning in robotics, heuristics usage in planning and long-horizon TAMP.

2.1 Modular robotics

The main characteristics of modular robotic systems include their physical structure and their degree of autonomy. In this section, we contemplate these aspects and give examples of existing modular robotic systems.

Modular robotic systems are typically divided into two categories based on their hardware architectures: *homogeneous* or *heterogeneous* [8], [11], [40], [41]. To this day, most research work focuses on homogeneous hardware architectures, which means that robot modules of a single type form connections. Notable examples of homogeneous modular robots are SWARM-BOT [10], Freebot [19] and hTetro [17]. In difference to that, heterogeneous modular robots are constructed from several different types of modules. Such systems can surpass the performance of homogeneous systems, as with different hardware types, more specialised parts can be used in such architectures. For instance, Thor is a modular robot that can easily navigate through rough terrain in one configuration and pick up or place a wide range of objects in a different configuration of its modules [42]. Extending the idea of heterogeneous systems are works where not only active, robotic structures are part of the configuration, but also passive parts. For instance Leder at al. [43] presented a modular robotic system for construction tasks, in which independent robot units cooperate by connecting themselves to wooden struts that also serve as building materials. Such a system enables simple and cheap robotic units to execute more complex motions compared to acting independently. However, forming heterogeneous connections proves to be much more challenging from a hard- and software perspective for reasons such as the need of a sophisticated docking system between the different parts, as well as controlling the kinematics of miscellaneous modules. Hence, it is little surprising that to our knowledge, there is no system to this day that is able to autonomously incorporate a wide range of different module types to form a single robot.

The degree of autonomy of a modular robotic system refers to how much assistance the system necessitates when acting and especially when reconfiguring itself [8]. In the context of TAMP, full autonomy signifies that a robot is able to plan all actions that are necessary to execute a certain task and subsequently find the right movements to execute the plan completely on its own. So far, only few works addressed the challenges related to fully autonomous TAMP for modular robots. Brunete et al. [44] presented a control architecture for modular robots in which low-level control is performed separately in each submodule and high level control over the system behaviour is exerted by a central control instance. This central instance is able to sense the configuration of the robot and choose an action from those available in the configuration. Despite experiments that demonstrated the efficacy of the control mechanism, a limitation of this work lies in the simplicity of the central action planning mechanism. The high-level action control of this work relies vastly on simple and predefined reactions to events in the environment, so task and motion planning is not combined in the sense of TAMP. Daudelin et al. [45],

presented a modular robot system, which is able indeed able to plan and perform actions autonomously. However, these capacities are limited by its central control mechanism, which requires all modules to stay in proximity of the central sensing unit. In addition, the system still uses closed loop control for behaviours, which further restricts its capacities to very simple tasks. In particular, the high-level task planner of this work is still not integrating a motion planner into the planning process, so yet again this system does not perform integrated TAMP. In fact, to this day, we are not aware of any modular robot system that uses combined TAMP to plan and execute a long sequence of actions autonomously in order to reach one or multiple long-term goals.

Another facet of modular robot autonomy is self-reconfigurability. The reconfiguration process ideally encompasses three stages: recognising that reconfiguration is necessary, determining an ideal robot configuration for a given task, and execute the reconfiguration plan. Autonomous reconfiguration remains one of the biggest challenges for modular robots and most of the previous works solely focuses one of those aspects. Some works systems deal with finding optimal configurations, but the presented robots need human assistance to execute the reconfiguration of the modules [16], [46], [47]. Other systems can self-reconfigure their modules, but only in a very simple way, like extending a chain [11], [21], [48] or grid [49], [50], which does not require a reconfiguration plan. None of these works addresses the challenge of *when* a modular robot should reconfigure its modules [16], [46], [47], [51], [52]. So far, we are only aware of one system, which performs all three steps of the self-reconfiguration pipeline autonomously [45]. As stated above though, this systems capacities are limited and it does not perform combined TAMP in the sense that we aim to do.

2.2 Combining task and motion planning

TAMP attempts to find a sequence of discrete actions and corresponding continuous motions that is needed to perform a certain task. A full review of TAMP algorithms can be found in [30]. In this section, we will give an overview over the works that are most relevant to the present work.

The configuration space of a robot is the space of all possible configurations across all robot parts [53]. A point in this space is a full specification of every degree of freedom of the robot. *Motion planning* of a given action is equivalent to determining a trajectory in configuration space from a start to a goal, constrained by the geometry and kinematics of the objects. To find such a trajectory, most modern integrated planners use either *sampling-based* motion planning methods [54]–[56] or *optimisation-based* motion planning methods [36], [57], [58]. In sampling-based motion planning, sample motions are generated and discrete search methods are used to a valid sequence of the samples. In addition, the feasibility of a sampled motion is evaluated by a collision detection mechanism. Optimisation-based motion planning on the other hand tries to optimise the trajectory in configuration whilst using constraints to model the physical properties of the environment and the robot kinematics in order to avoid collisions and physically impossible states of the robot. For instance, the motion planner that is used in LGP, as formulated by Toussaint [27], uses optimisation-based motion planning. An advantage of optimisation-based TAMP approaches is that it enables to directly optimise motion trajectories in a globally consistent manner, which, for instance allows to generate grasping

or handover motions efficiently [32].

Task planning, or symbolic planning, refers to finding an abstract action sequence called *skeleton* [59], whilst neglecting the geometrical details that the execution of the sequence entails. The execution of all actions in the skeleton leads from the initial logical state to a state satisfying all goal conditions. To find a valid skeleton, most integrated planners perform a search on the symbolic problem graph, where the vertices are symbolic states and the edges are the symbolic actions [27], [59]–[61].

The above methods for task planning and motion planning respectively, are standard approaches in the field of robotics. Consequently, the key differences between the different integrated TAMP planners usually do not arise from the choice of motion or task planning algorithm, but rather from the way how symbolic planning and motion planning are interacting during search. Most notably, *satisfaction-first* approaches differ from *sequencing-first* methods [30].

Satisfaction-first TAMP approaches call the motion planner before planning a symbolic action sequence. To do so, some motions and poses are precomputed in configuration space in order to use those quantities as discrete motion building blocks and perform tree search on the resulting discrete domain [62]–[64]. This has the benefit that the motion planner is called only a limited amount of times prior to task planning. However, those approaches come at the cost of restricting robot motions and poses to a discrete set. Thus, some solutions that are feasible in the original configuration space become infeasible in the discretised space [64].

The majority of current TAMP planners however perform *sequencing-first* planning [30], which makes use of the full configuration space. Such strategies begin planning with determining a valid symbolic action sequence first, and subsequently try to satisfy the constraints of this sequence during motion planning. Notably, LGP [27], first performs tree search over the space of symbolic action sequences and subsequently tests the symbolically-valid sequences for feasibility with an optimisation-based motion planner. On the other hand, Lozano-Pérez and Kaelbling [59] first perform tree search over the symbolic space as well, but subsequently check the feasibility of the sequence by discretising the geometric parameters of the action sequence, thus formulating the motion planning as a constraint satisfaction problem (CSP).

Such *sequencing-first* approaches have produced promising results in constrained environments like *pick-and-place* scenarios [26], [61], [64] and for *navigation among movable obstacles* [29], [65]. Recently, Hartmann et al. [36] were first to demonstrate the efficacy of LGP for long-horizon planning problem as well. However, it remains a main challenge for *sequencing-first* strategies that calls of the motion planner to evaluate the feasibility of an action sequence are a performance bottleneck [36].

2.3 Heuristics usage in planning

As robot task planning is usually search based [30], the main challenge for robot task planners remains the combinatorial complexity of the search space size that arises when planning tasks in big and unstructured environments. In fact, the number of nodes in the decision tree increases exponentially with the number of available actions as well as with the number of actors in the planning environment [32]. In parallel to task planning, integrated TAMP planners need to additionally verify the geometrical feasibility of a

plan. This procedure is very costly, as a motion planner has to be called each time a plan is verified. Hence, heuristics can be crucial in order to search the decision space in reasonable time. For example, consider the task of grasping the yellow object in Fig. 1. A correct solution would have the robot climb on the stair and the grasp the object. Nevertheless, the grasp action is logically valid at any time the gripper at the end of the robot arm is free to hold an object. But any logical decision sequence that starts with gripping the object will be geometrically infeasible, as the object is out of reach. Any optimal TAMP algorithm would recognise this as soon as possible and prune any such sequence to avoid costly plans to the motion planner. Next, we will outline some previous heuristic-based approaches to improve TAMP.

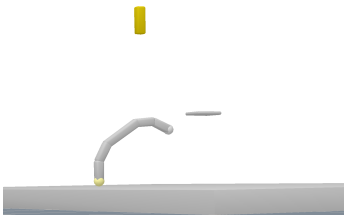

Figure 1: Using geometrical information, grasping the yellow object right away can be avoided.

A common heuristic approach is the incorporation of the Fast-Forward (FF) task planner [66] in TAMP. This planner makes use of a so-called *relaxed planning graph* (RPG) that ignores all negative effects of actions. The FF heuristic that the FF planner uses to pick the best successor state from the set of all geometrically valid candidates, is the number of actions that is necessary to reach a goal state in the RPG. Once the heuristic value of each state is determined, the enforced hill climbing algorithm is employed to find the best successor state for a given action. Garrett et al. [60] for instance used the FF planner in combination with the FF heuristic and several geometric tie breakers for states with the same heuristic value. To guide the heuristic search, they determined the reachability of objects with a sampling-based motion planner that they also used to plan the motions of the final plan. A disadvantage of this approach is that it still requires costly calls of the motion planner to assess reachability.

Learning-based methods attempt to predict the feasibility of a given action sequence and use these feasibility scores as heuristic values to guide symbolic search. For instance, Chitnis et al. [67] used reinforcement learning to learn promising actions from expert demonstrations. The so-learned policy values were then used as a heuristic to guide the search in the decision space. Yoon et al. [68] on the other hand used machine learning methods to learn a heuristic function that approximated the distance-to-goal for each state. Driess et al. [32] used deep learning in order to directly predict a feasible action sequence from initial scene images. If the initially predicted action sequences turned out to be infeasible, the predicted feasibility scores of individual actions were used as a heuristic to decide which node should be expanded next during symbolic search.

In order to limit the computational costs of calling the motion planner, Toussaint and Lopes [31] combined LGP with multi-bounded tree search. This method is based on the observation that the feasibility of a relaxed geometric problem is a necessary condition for the feasibility of the original motion planning problem, whilst being computationally simpler. In this framework, the outputs of the relaxed motion planning problems serve as a heuristic to guide the task planning. While the approach has produced some very promising results in [36], [69], the computation of the planning heuristic requires calling the motion planner to compute a lower bound to guide search. For big problem instances

this can still be problematic, as even the optimisation of relaxed geometric problems can take long time [36].

Other notable works use *geometry-based* heuristics, which leverage geometric information from the environment to improve planning performance, without calling the actual motion planner to compute a geometry-based heuristic score. Vega-Brown and Roy [70] for instance, made use of geometric abstractions to effectively plan tasks. In order to estimate the heuristic costs of executing an action, the authors defined upper and lower bounds of the costs for each predefined action. For example, the authors proved that in their framework, the costs of moving to an object can be bounded by the euclidean distance as minimal costs and the Hausdorff distance as maximal costs. In [71], the authors combined the FF planner with relaxed geometric reasoning to prune geometrically infeasible actions. Their geometric reasoning procedure evaluated several geometric conditions, such as reachability and stability, on a relaxed geometrical problem without calling a motion planner, while the actual motion planner was only called when the relaxed problem was feasible.

2.4 Long-horizon TAMP

Developing a TAMP solver which can plan long-horizon tasks remains a challenging problem [37], [72]. One technique to adapt TAMP algorithms to long-term tasks is to reduce search space size. We already described heuristic methods to achieve such a search space size reduction above.

Another technique to handle the challenges of long-horizon TAMP is iterative planning, i.e. problem decomposition. Such planners only execute and plan a portion of the task at each time step. The benefit of this approach is that very big problems are reduced to small-scale problems which are computationally tractable. Some previous works have tackled long-horizon TAMP by defining several sub-goals that have to be fulfilled in order to complete an entire long-horizon task. For instance, in [72] visual prediction models are used to predict sub-goal images that guide planning. In this work, the size of the problem fragments is determined by the number of sub-goals that should be met during planning. In contrast, limited horizon-based approaches define a fixed number of actions that are planned and executed, ignoring the semantics of the executed actions. Gupta et al. [73] predicted sub-goals from "play" data that are iteratively approached in order to find a long-horizon policy for task completion. Similarly, Castamann et al. [38] adapted the model predictive control, i.e. RHC algorithm, to TAMP. In each planning iteration a fixed number of actions was planned, but only the first action of each plan was in fact executed. In difference to that, Hartman et al. [36] predefined a number of actions that was planned and executed in each iteration of the planning algorithm. The latter work is most similar to the receding horizon LGP solver that we present in the present paper.

3 Task and motion planning for modular robots

The planner that we propose in the present work extends the basic LGP approach in order to perform TAMP for modular robots and consider geometric information while planning. The idea of LGP is to interleave state space search for task planning and optimisation-based motion planning. The state space search starts off with the initial symbolical state of the entire scene. The symbolic actions that the agents can perform are predefined in a standardised way, and called *decision rules*. To find a feasible action sequence, the planner searches the symbolic space by chaining actions in order to find a goal state. Once such a goal state is found, the motion planner is used to verify the geometrical feasibility of the plan. This is done using an optimisation-based motion planner. In this section, we will provide a detailed description of the planner that we used for modular robots.

3.1 Motion planning optimisation problem

Consider a robot and an object laying in front of it. Let the goal of the manipulation task be that the robot holds the object in its hand. For any such configuration space, the goal of LGP-based TAMP is to find a continuous and global path x from the initial state to the symbolic goal state, whilst fulfilling physical constraints. Let $\mathcal{S}$ be the symbolic state space, in which each state represents a high-level action sequence, e.g. $<$`(grasp BOX), (place BOX TABLE)`$>$, and a nonlinear trajectory optimisation program (NLP), that constrains the motions of the action sequence. To find a feasible solution of a TAMP problem, LGP uses tree search over $\mathcal{S}$ and solves the NLPs for individual states in $\mathcal{S}$ to guarantee the feasibility of the motions. Solving a NLP for individual symbolic states instead of exploring each possible motion trajectory in configuration space, keeps the complexity of a single optimisation problem low enough to effectively use non-linear optimisation techniques in high dimensional problems. Once a symbolic goal state is found and the corresponding NLP is feasible, the TAMP problem is solved. We will now describe the mathematical formulation of the optimisation problem that is used in the present work.

Let $\mathcal{X} \subset \mathbb{R}^n \times SE(3)^m$ be the configuration space of m rigid objects and n articulated joints with initial condition x_0. For a set of symbolic goals g, LGP tries to find a path $s_{1:K}$ in the decision tree, such that the last symbolic state in the path is part of the possible end states, i.e. $s_K \in \mathcal{S}_{goal}(g)$. The action sequence corresponding to $s_{1:K}$ is described by $a_{1:K}$ and called skeleton. In order to find the continuous motions, LGP tries to find a path $x : [0, KT] \to \mathcal{X}$ in configuration space, where $K \in \mathbb{N}$ is the total number of kinematic modes in the skeleton, also called phases, each of which is being held for time $T \in \mathbb{R}^+$. The exact values of K and T are subject of the LGP problem as well. Solving an LGP instance with goal g means to minimise

$$\min_{x_{1:KT}, s_{1:k}, a_{1:k}} \int_0^{KT} cost(x(t), \dot{x}(t_k), \ddot{x}(t_k), s_{k(t)})\ dt \tag{1a}$$

s.t.

$$x_0(0) = x_0,\ \widetilde{s_0} \tag{1b}$$

$$\forall_{t\in[0,KT]} : h_{path}(x(t), \dot{x}(t_k), \ddot{x}(t_k), s_{k(t)}) = 0 \quad (1c)$$
$$g_{path}(x(t), \dot{x}(t_k), \ddot{x}(t_k), s_{k(t)}) \leq 0 \quad (1d)$$
$$\forall_{k\in 1,\dots,K} : h_{switch}(x(t), \dot{x}(t_k), \ddot{x}(t_k), a_k) = 0 \quad (1e)$$
$$g_{switch}(x(t), \dot{x}(t_k), \ddot{x}(t_k), a_k) \leq 0 \quad (1f)$$
$$a_k \in \mathcal{A}(s_{k-1}) \quad (1g)$$
$$s_k \in succ(s_{k-1}, a_k) \quad (1h)$$
$$s_K \in \mathcal{S}_{goal}(g) \quad (1i)$$

where h_{switch} and g_{switch} in eq. (1e) & (1f) represent the constraints on the switches between the different kinematic modes in x, and (1c) & (1d) reflect the geometric and differential motion constraints on the path (that are imposed by the joint limits and collision avoidance). The function $cost$ returns path costs to the phase $k(t) = \lfloor t/T \rfloor$ of the motion. The path costs that we use is the sum of squared accelerations plus *homing* costs. In order to solve the problem, the path x is assumed to be globally continuous and twice continuously differentiable within each kinematic mode. The discrete logical transitions between s_{k-1} and s_k are defined as a function $succ(\cdot, \cdot)$ of the symbolic state s_{k-1} and the action a_k at phase k. For each state s_k, the set of possible actions is defined by $\mathcal{A}(s_{k-1})$. The initial continuous state is defined in (1b) as x_0 and the corresponding initial symbolic description of this state is defined as $\widetilde{s_0}$. The TAMP problem is solved if all the above conditions are met and the set of goals g is a subset of the last symbolic state s_K of the skeleton.

To solve the individual NLPs, we use the *k*-order motion optimisation (KOMO) framework [74].

3.2 Multi-Bound Tree Search

The symbolic task planner of LPG performs tree search in the state space. To perform symbolic planning, there is a predefined set of symbolic actions, called *decision rules*. For each decision rule there are pre and post conditions that must hold before and after executing that action. Those conditions are defined using symbolic and kinematic predicates.

The symbolic predicates, e.g. (on X Y), describe the symbolic state of the system and restrict the available actions of the planner, based on general assumptions. For example, a gripper that holds an object cannot grasp a second object at the same time. Furthermore, when a robot performs a certain action, its motions are constrained by its kinematics and the laws of physicality. To specify the motion constraints of an action we use kinematic, i.e. optimisation-related, predicates that represent conditions, which should hold after the execution of a given action. The combination of all motion constraints imposed by an action sequence forms the corresponding NLP. Thus, during symbolic search, each action sequence node can be tested for feasibility by simply solving the NLP that is associated with that node. Each leaf node in the decision tree either refers to a symbolic state in which no further actions are applicable, or to a state in which all goals are fulfilled. In the latter case, the corresponding NLP is a candidate for solving the overall TAMP problem. In order to reduce the number of calls to the motion

optimiser, only these candidate NLPs will be subject to motion optimisation. If any such candidate NLP is feasible, the TAMP problem is solved. Thus, the symbolic planning in LGP performs a search over the state space in order to find one or several feasible NLPs. However, the number of candidate NLPs is generally too high to solve the complete problem specified by eq. (1) for each of them in an adequate amount of time. To alleviate this problem, we instead first solve relaxed versions of eq. (1) in order to assess the general feasibility. This so-called *multi-bound tree search* (MBTS) was introduced by Toussaint and Lopes [31] and it is based on the observation that the feasibility of each relaxed problem is a necessary condition for the feasibility of the original NLP, whilst being computationally simpler. As the feasibility of each NLP depends on the feasibility of the relaxed NLPs, those computationally simpler problems are also called lower bounds [31]. The three bounds that we use are called $\mathcal{P}_{seq}$, $\mathcal{P}_{path}$ and $\mathcal{P}_{seqPath}$.

The sequence bound, $\mathcal{P}_{seq}$, is the coarsest bound we use. The optimisation problem corresponding to this bound optimises the motion over the full action sequence, but with a very coarse time resolution. It only includes the switching poses, i.e. the path x from eq. (1) is discretised to a slice before and one after each kinematic switch across a given action sequence. This bound helps to prune all action sequences that contain invalid kinematic states. The next evaluated bound is $\mathcal{P}_{seqPath}$. This level is the full path optimisation problem with very fine time steps. In addition to the constraints from the problem in eq. (1), this bound constrains the motion to include all switching poses that were computed in the solution of $\mathcal{P}_{seq}$. If $\mathcal{P}_{seqPath}$ is feasible, the action sequence can be executed without any collisions. However, as the restriction to include the poses from $\mathcal{P}_{seqPath}$ can be a very limiting constraint, we use a third bound named $\mathcal{P}_{path}$ bound. The $\mathcal{P}_{path}$ bound problem is only evaluated if a sequence is logically feasible, $\mathcal{P}_{seq}$ is met, but $\mathcal{P}_{seqPath}$ is not met. $\mathcal{P}_{path}$ is the full path optimisation problem with very fine time steps. However, this bound does not restrict the motion to include the poses from the $\mathcal{P}_{seqPath}$ solution. Nevertheless, if $\mathcal{P}_{path}$ is feasible, the action sequence can be executed without any collisions.

A sketch of the full LGP algorithm using MBTS and the heuristics interface that we present can be found in Alg. 1. The details of it are explained in the upcoming section, where we describe how we integrate heuristics in MBTS.

3.3 Heuristics-enriched LGP

A key novelty of our approach is that the symbolical planner is aware of geometric information from the environment. To this end, we introduce a problem independent interface for heuristics into the LGP task planner. Using the interface, each node can be assigned a symbolic cost-to-go value that is approximated with the help of geometric information from the initial kinematics of the environment. A low heuristic cost-to-go value indicates that extending the action sequence will likely result in a feasible TAMP solution. The interface we propose allows to assign each state in the decision space a cost-to-go value, based on the last action in the corresponding decision sequence. These heuristic costs are then used to pick the next nodes for expansion or optimisation during planning. The heuristics-based task planning involves two components: *action specific heuristics*, which incorporate geometric information to estimate the costs of performing a given action in a certain state, and the *heuristics interface function* which integrates the

action specific heuristics into the classic LGP planning. Both components are described next.

Algorithm 1: The LGP algorithm with the heuristics interface. Heuristics-related contributions are in pink. MBTS-related contributions are in blue.

```
1  // the LGP loop
2  numSol ← 0
3  solutions ← ∅
4  while within time budget ∧ numSol < numNeededSols do
5      // explore decision tree
6      if heuristics are used then
7          node ← choose best node from expandQueue based on cost–to-go value
8      else
9          node ← expandQueue.getFirst()
10     for every child c of node do
11         if c is terminal then
12             for every node n on the path from root to c do
13                 P_seq.queue.append(n)
14         else
15             expandQueue.append(n)
16
17     // optimise motions of candidates
18     numSol ← size(solved.queue)
19     optBestOnLevel( P_sequence, P_seqPath.queue )   // selected based on heuristic
         value
20     optBestOnLevel( P_seqPath, solved )
21
22     // store solutions
23     if size(solved.queue) > numSol then
24         solutions.append(solved.queue.last())
25 return solutions
26
27 // optimise most promising motion on a level
28 Function optBestOnLevel(currentBound, nextQueue):
29     bestCandidate ← currentBound.queue.getBest()
30     solveNLP(bestCandidate, bound=currentBound)
31     if !feasible(candidate) ∧ currentBound = P_seqPath then
32         solveNLP(bestCandidate, level=P_path)
33     if feasible(candidate) then
34         nextQueue.append(candidate)
```

An interface for heuristics

The classic LGP approach [27], [31], already performs symbolic planning in an A*-like manner. So far however, only the costs of the NLPs were considered to guide search.

This approach is not very effective in large settings, as it explores action sequences that are likely to be geometrically infeasible, such as grasping objects that are out of reach. Hence, we introduce an LGP-interface for heuristics that can be used to assign each state in the decision space a heuristic cost-to-go value, based on geometric information. This interface allows the programmer to define a problem specific interface function upon initialisation of the LGP problem. To use the interface, the programmer has to specify a *cost-to-go function*, which computes the heuristic cost-to-go value for an input node. Note that if no heuristic is defined, the planning is performed in classic LGP fashion, using the bounds we described above. A simple example of a cost-to-go function would be to assign a very high cost-to-go value to any decision node that corresponds to grasping an object, which is at a greater distance to the robot than the reach of the robot gripper, in order to prune this node from the tree. Whilst the cost-to-go function that is passed to the LGP planner should be problem-specific, the interface in itself is problem independent, meaning that it can be used for any LGP problem.

Planning using the interface

If a cost-to-go function is defined, it is used in two steps of the logic-geometric planning: choosing which node to expand next and choosing which nodes' NLP to optimise next. The adapted algorithm using the symbolic cost-to-go values can be seen in Alg. 1. The contributions of the present work are highlighted in pink. In this algorithm, the LGP loop runs until the desired number of solutions for the motion planning problem has been found or until the time budget is maxed out. Inside the loop, new candidate nodes are generated by expanding a node from the *expandQueue*. This queue contains all nodes that have been discovered, but not expanded yet. Whenever a new node is expanded during search, the node with the lowest symbolic cost-to-go value is popped from *expandNext*. To perform this get operation as efficient as possible, a nodes symbolic cost-to-go is stored upon pushing it to the *expandNext* queue, and then used when popping a node from that queue at a later point in time. As you can see in line 9, if the heuristics interface is not used, the first value from the queue is popped instead. Consider a given decision sequence $s_{1:k}$ of length $k \in \mathcal{N}^+$. During node expansion, all actions that are applicable in state s_k are examined. If the sequence including such a new action ends in a logical goal state, the node and all its predecessors are pushed to the $\mathcal{P}_{seq}$ queue for sequence bound motion optimisation. Otherwise, the node is pushed to the *expandQueue*. After node expansion, the best nodes from the $\mathcal{P}_{seq}$ and $\mathcal{P}_{seqPath}$ queues are popped, based on motion optimisation costs, and their respective NLPs are solved. If the NLPs of these nodes are feasible, the lower bounds of the optimisation problems are met and the nodes are pushed to the next queue. In the case of $\mathcal{P}_{seq}$, the next queue is $\mathcal{P}_{seqPath}$ bound queue. If a node does not meet the constraints of $\mathcal{P}_{seq}$ bound, it is removed from the search tree. If a node fails to meet the $\mathcal{P}_{seqPath}$ bound, this bound is relaxed and satisfying the $\mathcal{P}_{path}$ bound becomes a sufficient condition for feasibility (line 32). Next, if any node met its $\mathcal{P}_{seqPath}$ or $\mathcal{P}_{path}$ bounds, a new solution has been found and it is added to the list of solutions. As soon as a predefined number of solutions has been found, the algorithm stops.

Action-specific heuristics

The heuristic cost-to-go function can be defined by the programmer based on general and problem specific information. To support programmers in the definition of such a cost-to-go function, we introduce action-specific heuristics that can be used as building blocks to create heuristic cost-to-go functions following a *mix-and-match* principle. Given a symbolic action sequence, such a cost-to-go function can simply check the type of the last action in the sequence and return a cost-to-go value, based on the action-specific heuristic. Depending on the actions that are available to the planner, the action-specific heuristics can simply be chained to form a problem specific heuristics interface function. In order to leverage geometric information during symbolic planning, all action-specific heuristics that we present use the same source of geometric information. Since calling the motion planner during task planning to obtain the actual object positions is too expensive, we use the initial kinematic state of the environment. This idea is based on the assumption, that each environment contains immobile objects, such as the floor or stairs that the mobile objects are standing on. If these base objects are in fact immobile, the position of the base object that a robot is standing on can be used as an approximation of the actual robot position. Based on this geometric information, the symbolic planner can avoid infeasible actions like grasping the target in Fig. 1, because the target is too high from the current base object. The principle is illustrated in Fig. 2. Here, the distance of the target to the base is larger than the length of the extended crawler, from which it follows that an immediate grasp will be infeasible. Next, we will describe all action-specific heuristics that enable leveraging geometric information during task planning. An integrative summary of all heuristics can be found in the Appendix.

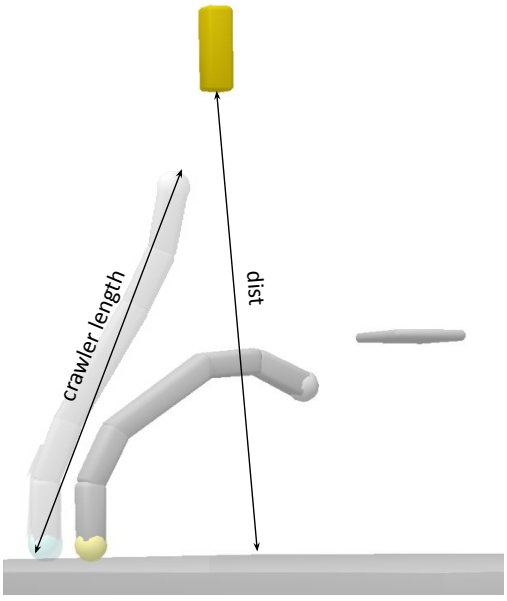

Figure 2: The reachability check we use in our heuristics.

The *grasp heuristic* checks if the object that is to be grasped is within reach. To check if the target object is within reach, the heuristic checks if the euclidean distance between the base object that the robot stands on and the position of the target object is not larger than the reach of the robot module. If the distance is in fact bigger, the heuristic returns a heuristic value of 1000, thus effectively pruning this state from the search tree. However, if this is not the case, the object is considered close enough to try and test grasping. In this case, the heuristic returns action costs of 0. As no action sequence including the grasp of an object that is out of reach will be a solution of the

TAMP problem, pruning these action sequences improves symbolic search.

Similarly to the grasp heuristic, the *place heuristic* checks if the object on which another object shall be placed is within reach. This check is based on the euclidean distance between the base that the manipulator stands on and the placement region. If the target tray object is out of reach, the node is pruned from the decision tree by returning costs of 1000. Otherwise, the returned costs are 0. This heuristic improves search for the same reason as the grasp heuristic.

The *step heuristic* checks if the object that the robot is standing on before the step is within reach of the target base object. To do so it considers the euclidean distance between both objects. If this distance is larger than the reach of robot, the step action is geometrically infeasible and the heuristic prunes the node. If the target base object is indeed within reach, the heuristic returns the amount by which the distance of the robot to its target of navigation will be reduced by the step. Thus, if a step is made towards the target of navigation, the heuristics returns negative actions costs. If a step is in fact moving away from the target of navigation, the heuristic will return positive action costs. This way, the step heuristic can be used very effectively for navigation planning. If the initial heuristic costs is the initial distance to the target of navigation, the heuristic of each step can simply be added to the initial distance.

The *disconnect heuristic* checks if the prior decision in the action sequence was to connect the same modules that should now disconnect. If this is the case, the heuristic prunes the current node from the search tree, as such a decision sequence will never be optimal. Since the connect action lifts one endeffector from the ground and the disconnect action puts that effector back down again, subsequent connect and disconnect actions can have the same geometric outcome as taking a step. However, in this case, the two actions connect and disconnect can be replaced by a single step action. If the connect and disconnect subsequence has no geometric effect on the other hand, the geometric state of the robots is the same as prior to executing the two actions. In this case any feasible decision sequence excluding the connect and disconnect subsequence will be feasible as well, but shorter.

In similar vein, the *connect heuristic* checks if the prior decision in the action sequence was to disconnect the modules that should be connected now. If this is the case, the heuristic prunes the current node from the search tree, as such a decision sequence will never be optimal for the same reason as we stated for the disconnect heuristic above. Furthermore, the heuristic checks if the two robot modules that are to connect, are within reach of each other. This check is performed with the same reachability check that is also performed by other heuristics, e.g. the grasp heuristic. Since we previously demonstrated that action sequences including subsequent connect and disconnect action are not ideal and the heuristic further prunes geometrically infeasible solutions, it follows that this heuristic improves symbolic search.

In addition to these action-specific heuristics, problem specific information can be used to improve the planning results. In the experiments section, we will describe the specific cost-to-go functions, which combine the action-specific heuristics with other information.

3.4 Symbolic action templates for modular robots

To perform symbolic planning, we define symbolic actions, called *decision rules*. Each of those actions is defined based on symbolic and kinematic predicates. Whilst the symbolic predicates have the purpose to aid symbolic planning, the optimisation predicates specify kinematic constraints that should or should not hold after the execution of a specify action, thereby constraining the corresponding motion. In this section we describe the decision rules and how we defined the motion constraints.

(a) A single crawler module

(b) A mobile base plate module

(c) Two connected crawler modules

(d) A crawler connected to a mobile base

Figure 3: The robot modules used in the present work and their configurations.

The robot modules

For the present work, we focus on two types of robot modules with different capabilities. Robot modules of both types are able to act as independent robots. Furthermore, the robot modules can be connected to form a bigger robot structure. The two robot module types are depicted in Fig. 3 and are:

(i) A crawler-like robot, similar to the one introduced in [69]. Each crawler module is composed of 7 joints and has two endeffectors. The endeffectors have a sphere shape and can be used to provide stable support or to function as a gripper. Thus, it can pick and place objects and perform a form of bipedal locomotion.

(ii) A mobile base plate that is able to move freely in the xy-plane. The base plate has the shape of a box. It is a single object and cannot perform any action, but move around. At the top of the base plate is a marker object, which indicates where the crawlers can connect to the base plate.

Action optimisation constraints

We use seven predicates to constrain path optimisation. The *touch, stable and above* predicates have been used in previous works [31], [36] and their specification can be found in Table 1. Similarly, the *touchBoxNormalX* predicate has been used in previous LGP problems as well. It constrains the distance of object X and Y to be 0, the Z-axis of the gripper to face upwards and the X-axis of the box object to be perpendicular to it. The *connectCrawlers* is a predicate we designed specifically to work with the

crawler robots that we described above. Given two crawler endeffectors A and B, this predicate constrains their distance to 0, making them touch. Furthermore, to simulate a realistic hardware connection, the two endeffectors are constrained to oppose each other. This is equivalent to restricting the z-vectors z_A and z_B of the endeffectors' local coordinate systems to oppose, i.e. to be anti-parallel. Using a property of the dot product of two anti-parallel vectors and the fact that the z-vectors can be assumed to have unit length, we obtain the optimisation constraint of $z_A \cdot z_B = |z_A||z_B| = -1$. In addition, the endeffectors are constrained to have their coordinate systems aligned, as this leads to more realistic solutions. With the z-axes already being constrained by their anti-parallelity, only one more pair of axes needs to be aligned to align the coordinate systems. We chose the x-axes of both endeffectors. Let x_A be the x-axis of the A's local coordinate system and let y_B and z_B the y- and z-axes of B's local coordinate systems. The optimisation constraints that are used are $x_A \cdot y_B = 0$ and $x_A \cdot z_B = 0$.

Predicate	Constraints
[touch X Y]	the distance between X and Y equals 0
[stable X Y]	create a stable joint from X to Y, constrained to zero velocity
[above X Y]	X is above Y with a distance of > 0
[connectCrawlers X Y]	the distance between the endeffectors X and Y is zero the z-vectors of the endeffectors oppose each other, the other two axes are perpendicular
[touchBoxNormal X Y]	the distance between X and Y equals 0 the z-axis of object X points upwards the z-axis of object X and the x-axis of object Y are perpendicular

Table 1: Predicates to constrain trajectory optimisation.

Single module actions

The individual robot modules are able to perform three actions when not connected to other modules. The first two of those actions are *grasp* and *place*, which were already used in a similar way in [32]. A thorough description of the pre and post conditions of those, and all other actions can be found in the Appendix. As the name suggests, the action `(grasp A X)` makes a crawler robot with free gripper endeffector A grasp the object X. The action `(place A X Y)` is the logical counterpart of the grasp operation and it makes the robot place the object X from its gripper A onto the object Y.

The third single robot action, *step*, is specific to the crawler robot. Assume a crawler robot with endeffectors A and B that stands on object X with its effector A. Executing `(step A B X Y)` makes the crawler take one step from object X to object Y, changing its supporting end from effector A to B. In the process of stepping, the root of the kinematic chain of the robot is inverted from A to B. In order to limit the combinatorial possibilities of objects to move from and to, we defined the logical class of *base* objects.

This class contains all objects that the robot can walk on. Objects that we excluded from that class are for instance other crawler robots in the scene and objects that are too small to provide stable support.

Action	Optimisation predicates
`(step A B X Y)`	[stable Y B] [touch Y B]
`(grasp A X)`	[stable A X] [touch A X]
`(place A X Y)`	[above X Y] [touch X Y] [stable Y X]
`(connect A B C D)`	[stable C B] [connectCrawlers C B]
`(disconnect A B C D X)`	[stable X A] [touch X A]
`(stepTogether A B C D X Y)`	[stable Y D] [touch Y D] [touch C B] [stable C B]
`(connect2mobileBase A B X Y)`	[stable Y B] [touchBoxNormalX Y B]

Table 2: Decision rules and the motion constraints that they imply.

Multi-module actions

Multi-module actions involve two or more separate robot modules. For the present work, we focused on two crawler robots chaining themselves to form one big crawler robot and crawler robots attaching themselves to a mobile base module to form a mobile manipulator. Both modular robot configurations can be seen in Fig. 3.

Assume two crawler robots with endeffectors A, B and C, D respectively, where A and D are the stable links of the modules with any base object. There is a kinematic chain from a base object via A to B and another kinematic chain from a (potentially different) base object via D to C. Executing the `(connect A B C D)` action connects the two modules at their grippers B and C and lifts A, so that D is the support of the robot on the base object, and A at the other end of the kinematic chain of the connected modules. Thus, upon connecting, a new crawler robot with twice the length of a single crawler module is formed, which can use its gripper to perform the *grasp* and *place* actions just like a single module. To perform this action, the kinematic chains of the two modules are joined, resulting in a single long kinematic chain, where there is a stable joint from B to C. Furthermore, the *connectCrawlers* motion constraint is used in order to ensure a proper connection between the grippers is formed.

The logical counterpart of the *connect* action is the *disconnect* action. Assuming a crawler robot, comprised of two modules just as described above, executing `(disconnect A B C D X)` breaks the kinematic chain of the assembled robot back into two chains and puts A on base object X, where a stable link from X to A is formed. The action results in one module with a free gripper B and a stable base link at endeffector

A, and another module with free gripper D and stable base link at endeffector D. Consequently there are two kinematic chains: one from X via A to B and another kinematic chain from a base object via D to C.

Similar to the *step* action for a single crawler module, a crawler comprised of two modules can take steps as well. However, for this action the kinematic chain has to be inverted with each step, which proves to be challenging if several crawler modules are chained. Thus, we created a separate action for this case called `(stepTogether A B C D X Y)`. In similar vein to the step action of a single module, the step is from base object X to base object Y. However, this time, there are four robot effectors involved. Assume that the crawler modules are connected at effectors B and C and that there is a kinematic chain from endeffector D via endeffector A to the base object X. The `(stepTogether A B C D X Y)` action creates a stable link between base object Y and endeffector D and inverts the kinematic chain of the entire crawler such that D becomes the end of the chain and the link to base object X is broken. To do so, the action imposes a stable connection optimisation constraint from Y to D and it flips the stable connection constraint between the effectors C and B. Furthermore, it imposes touch constraints between D and Y as well as C and B.

A second type of modular robot configuration that we used for the present work is a mobile manipulator comprised of a mobile base plate and one (or more) crawlers mounted onto it. In order to create such a configuration, the `(connect2mobileBase A B X Y)` can be used. Given a crawler with endeffector A standing on base object X and B as the other endeffector, this action creates a stable joint from mobile base Y to endeffector B and breaks the link between A and X. Furthermore, it constrains the connection motion with the [touchBoxNormalX B Y] predicate, s.t. the crawler is attached perpendicularly to the mobile base. Once a mobile manipulator is created, the integrated crawler module can execute all actions it could before, apart from walking as its one endeffector is attached to the mobile base. However this type of movement is not necessary, as the base can be used for movements. Thus, the crawler can configure itself to another crawler to extend its reach or perform grasp and place manipulations.

3.5 Receding horizon-LGP

In order to solve long-horizon tasks in dynamic environments, we extend the presented framework framework with a receding horizon LGP solver that we call RH-LGP. The RH-LGP planner uses a predefined number of actions as a planning horizon and iteratively plans and executes motions for that horizon. Once the horizon is reached, the optimisation time window is shifted, and the next motions are planned until the final goal is fulfilled. The planning process is outlined in Fig. 4.

Let $n \in \mathcal{N}^+$ be the horizon length. The iteration count is denoted by $t \geq 0$. At $t = 0$, the RH-LGP planner takes the initial kinematics of the environment and the set of goals as input. Now, the standard LGP solver with geometry-based heuristics is used to plan the first n actions. In each iteration, task planning is deemed successful if all symbolic goals are met. This means that the task planner always solves the full problem of finding an action sequence that leads from the current symbolic state to a goal state. Once a symbolic solution with ℓ actions has been found, the horizon n is used for motion planning: In the motion planning step, the motions of the first $\min\{\ell, n\}$ actions are

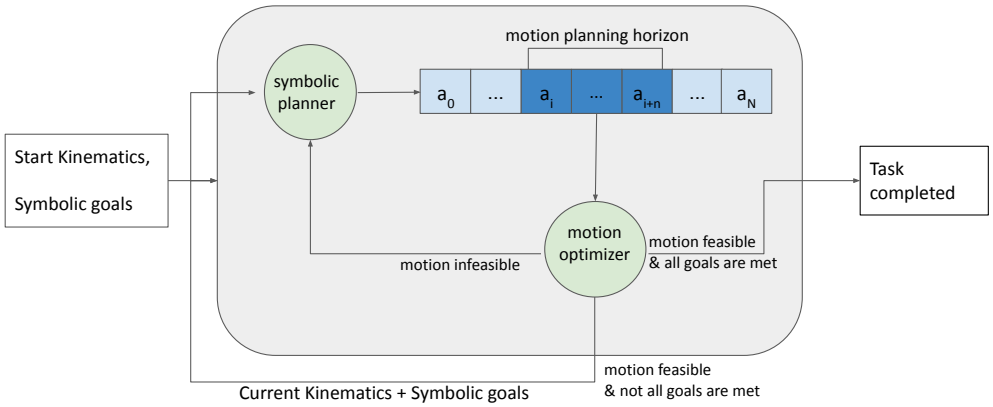

Figure 4: The RH-LGP planner that we used for iterative planning.

considered as solution candidates and the corresponding motions are optimised using the MBTS procedure we outline in Alg. 1. Once motion optimisation is successful, the actions are executed. If the corresponding action sequence meets all goals, a successful plan has been found and executed at this point, so the planning process ends. If this is not the case, the next iteration $t+1$ of the RH-LGP planner starts. The input of the planner in iteration $t+1$ is the global goal, and the kinematic state of the environment after iteration t, which we also call the current kinematic state. In order to use the LGP planner within RH-LGP, we made adjustments to it that we will discuss next.

First of all, we initialise each LGP tree by adding the LGP nodes of all previously executed actions. If for instance, the previously executed actions are `<(grasp BOX),(place BOX TABLE)>`, then the LGP tree will be initialised to the LGP tree depicted in Fig. 5. The node that will be expanded next in that scenario is the leaf node of this path, i.e. the last action taken by the robot, in the example the `(place X Y)` decision node. This way, we ensure that the logical state stays consistent with all actions that have previously been executed.

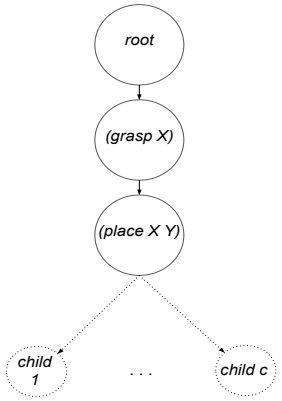

Figure 5: A newly initialized LGP tree after a previous RH-LGP iteration.

In difference to the standard LGP solver, the iterative RH-LGP planner only evaluates the motion constraints of the next n actions (or less, if the overall goal can be achieved faster). Despite using the kinematic state of the system after executing the previous set of actions, some optimisation constraints of earlier windows have to be carried along to ensure motion optimisation feasibility. In iteration 0 this is obviously not necessary, so we only describe the procedure for $t > 0$. First of all, consider the skeleton of a symbolic task solution that we find in iteration t. All skeleton

entries from earlier iterations $i < t$ are added to the current skeleton iff. their timing overlaps with the current optimisation window w_t. In addition, motion smoothness at the transition between iterations is guaranteed by including the constraints that hold in the last time step of the solution of $t-1$ as initial motion constraints in iteration t.

In addition to using optimisation constraints from previous iterations, the kinematic endstate of iteration $t-1$ serves as initial kinematic state of the iteration t (for $t = 0$ the global initial state is used). To ensure realistic motions, one modification is made to each initial kinematic state before the LGP problem is solved: When a kinematic predicate such as [stable X Y] enforces a stable link between X and Y, LGP creates a *free joint* between the two objects. Since these *free joints* restrict the position of the objects its ends only with additional constraints, we convert all *free joints* in the kinematic endstate of $t-1$ to *rigid joints* to ensure that linked objects that remain visibly linked in iteration t.

4 Experiments

In order to illustrate the capabilities of our long-horizon planner, we conducted several experiments in different scenarios. All experiments were conducted in simulation, using the robot modules we described previously. Example videos of the tasks can be found at `https://www.youtube.com/playlist?list=PLpZ8d_Qv_8kiL0rBzWqJRWbPIO1zUUZX5`.

4.1 Using heuristics for modular robot manipulation planning

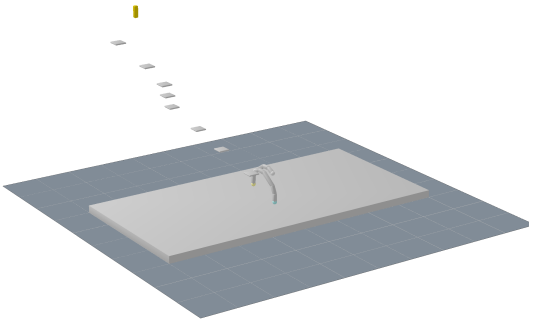

Figure 6: The scenario used for the first two experiments.

To measure the impact of heuristic functions on planning, we considered the task of grasping a distant object and compared the performance of the planner using the novel heuristics interface with the performance of the classic LGP planner. In this scenario, the position of the target was out of reach for both crawlers, and stairs had to be climbed to get within reach of it (Fig. 6). Furthermore, the position of the stairs was randomised. We ensured that each random problem instance was feasible by defining coordinate ranges for each stair step. A solution for two crawlers in this task is that they connect and then climb the stairs to reach the target. We measured the impact of heuristics on planning by comparing the planning performance with usage versus the planning performance without usage of the heuristics.

n	Heuristics				No Heuristics			
	Time in s.	Tree nodes	Expanded nodes	Solution len.	Time in s.	Tree nodes	Expanded nodes	Solution len.
1	0.33	21	3	2	2.31	65	8	2
2	3.83	111	17	7	7.98	211	22	2
4	5.38	190	22	8	350.16	2746	591	5
8	7.25	309	23	9	–	–	–	–
16	35.76	685	29	12	–	–	–	–
32	309.43	2488	53	20	–	–	–	–
64	470.58	4088	53	27	–	–	–	–

Note: scenarios without metrics could not be solved at all

Table 3: Planning performance using with usage of heuristics and without.

The problem specific cost-to-go function that we used can be seen in the Appendix. It contains of the step, connect, disconnect and grasp action heuristics. In addition, we used the problem specific knowledge that connecting is probably beneficial and discounted the heuristic values of the connect action heuristic by 1.5. As initial heuristic cost-to-go value we used the euclidean distance between the target object and an endeffector of

the crawlers. With the execution of each action, the cost-to-go was updated based on the action heuristic values.

The results of the simulation can be seen in Table 3. We compare the planning times for several numbers of stairs in the scene in order to assess the scalability to real world scenarios of our approach. For all runs, we find that planning with the heuristic function was faster. In addition, with 8 or more stairs in the scene, no solution could be found without using the heuristics interface. The action sequences that the planner was able to produce contained up to 27 actions.

n	One crawler				Two crawlers			
	Time in s.	Tree nodes	Expanded nodes	Solution len.	Time in s.	Tree nodes	Expanded nodes	Solution len.
1	0.09	10	2	2	0.33	21	3	2
2	0.85	57	13	5	3.83	111	17	7
4	0.99	85	13	7	5.38	190	22	8
8	11.39	181	17	10	7.25	309	23	9
16	26.29	505	27	16	35.76	685	29	12
32	423.16	4421	129	43	309.43	2488	53	20
64	–	–	–	–	470.58	4088	53	27

Note: scenarios without metrics could not be solved at all

Table 4: Planning performance depending on the number of available modules.

4.2 Single-module crawler versus two-module crawler

In this experiment, we used the same setting as in the previous task. In order to investigate the potential of modular robots, we compared the planning performance having only one crawler module in the setting versus having two crawlers in the scene. In the latter setting, the two crawlers could connect, but were initially unconnected. We again used the same heuristic function as in the first experiment. The results can be seen in Table 4. With several robot modules in the environment, the planner was able to solve a scenario that it could not solve with just a single module crawler in the scene. Planning times were similar for both conditions. The longest action sequence that the planner was able to plan included 43 actions.

4.3 Planning tasks involving several different actions

In order to evaluate our approach in a more complex and realistic scenario, we modified the task we used previously. There were two crawler robots in the environment. The position of the target was still out of reach for both crawlers and stairs had to be climbed to get within reach of it (Fig 7). However, in this modified setting, the crawlers initially stood far away from each other and had to walk towards each other in order connect. Furthermore, there were multiple goals: two target objects had to be held simultaneously. In order to facilitate navigation on the ground, i.e. the lowest base object, we discretised the ground into several parts. This was implemented by splitting the ground object into three separate objects as it can be seen in Fig. 7.

n	Time in s.	Tree nodes	Expanded nodes	Solution len.
1	23.13	883	117	13
2	31.18	1293	157	13
4	19.35	671	67	11
8	28.35	1157	77	12

Table 5: Planning a complex scenario for different numbers of stairs.

The cost-to-go function we used for this setting can be seen in Alg. 3. It includes the step, grasp, connect and disconnect action heuristics. As initial cost-to-go value we used the sum of the initial distance between the two crawlers and the minimum initial distance between the target and any crawler. The results of planning this scenario for different numbers of stairs can be seen in Table 5. The longest action sequence that was found by the planner included 13 actions. All solutions included *step, connect, stepTogether, disconnect, and grasp* actions.

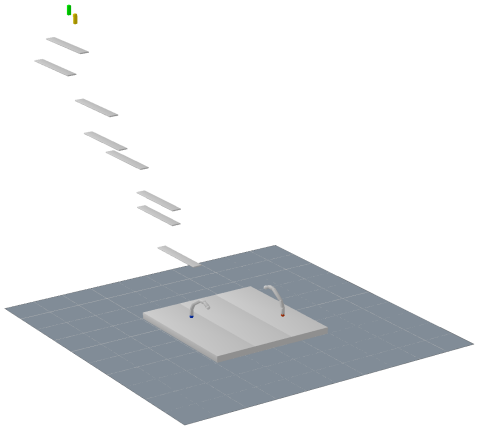

Figure 7: The scenario used for the third experiment.

4.4 Pick and place task involving modular mobile manipulator

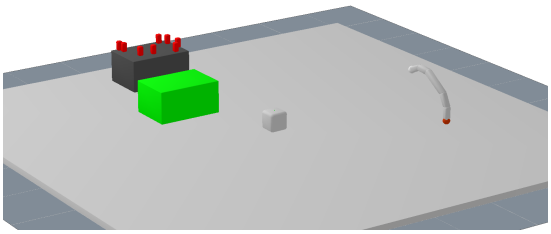

Figure 8: The scenario used for the mobile manipulator experiment.

In this scenario, we planned a task that involved another configuration of the robot modules. We used one crawler module and one mobile base plate module. The initial environment can be seen in Fig. 8. The task was to transfer n objects from their initial position to a target tray. Once a crawler is holding an object it cannot walk, thus the solution of this problem involved that the crawler module connects to the mobile base, forming a mobile manipulator. When this configuration was assumed, the remaining pick and place task plan could be executed. In Table 4, we compare planning times for different numbers of objects that needed to be placed.

The heuristic function we used can be seen in Alg. 4. It includes the place, grasp and disconnect action heuristics. The initial cost-to-go value we used was the number of

objects that needed to be transferred from the initial to the target tray. The results can be seen in Table 6. The longest action sequence that could be planned was 17 actions long.

n	Time in s.	Tree nodes	Expanded nodes	Solution len.
1	0.97	7	2	3
2	2.67	12	4	4
4	6.46	25	8	9
8	45.61	68	19	17

Table 6: Mobile manipulator pick-and-place task for different numbers of objects.

4.5 Iterative planning of the previous tasks

In this experiment, we planned the previous four tasks with the RH-LGP solver. For each task, we first solved the scenario with the highest number of objects with the RH-LGP, and in addition we solved one scenario that could not be solved by the normal LGP solver we used for the experiments above. All tasks were planned using the same heuristic cost-to-go functions that we previously used to plan them in the experiments above (see Appendix). The results of the experiments can be seen in Table 7. The setting of two crawlers climbing 64 stairs could not be solved by RH-LGP. In the following section we discuss the implications of these findings.

Task (experiment n°)	n	Horizon	Time in s.	Expanded nodes	Solution len.	tractable for LGP
One crawler climbing (1)	32	6	273.61	353	33	yes
One crawler climbing (1)	45	8	391.04	283	49	no
Two crawlers climbing (1 & 2)	32	3	362.40	274	13	yes
Two crawlers climbing (1 & 2)	64	–	–	–	–	yes
Complex climbing (3)	8	4	72.62	286	14	yes
Complex climbing (3)	25	8	95.97	154	21	no
Mobile manipulator (4)	8	4	5.87	53	17	yes
Mobile manipulator (4)	16	4	20.62	172	33	no

Table 7: Performance of the RH-LGP solver in various tasks.

5 Discussion

This paper presents a LGP-based long-horizon TAMP solver for modular robots including a problem-independent interface for heuristics, and an iterative LGP version that we call RH-LGP.

5.1 Modular robots facilitate task planning

In the second experiment that we conducted, we found that reaching the object that is 64 stairs high with only a single crawler robot was computationally intractable, despite a solution existing. On the other hand if two crawlers are available, the same task is computationally tractable, because the solution has fewer steps than a possible solution involving a single crawler. In fact, a two-module crawler needed less actions to solve the 64 stairs task than a single module crawler needed to solve the 32 stairs task. Hence, our results demonstrate that using modular robots can be useful for solving manipulation planning tasks. This finding goes beyond the usually described benefits of modular robotics, which are robustness, low-cost and versatility [8], [18].

5.2 Comparison with other modular robotic systems

Previous works demonstrated that heterogeneous modular robots can surpass the performance of homogeneous architectures, but most of them are not self-reconfiguring [9], [16], [42]. The system that we present, however, is able to plan motions and actions for different types of robots and execute them fully autonomously. In our experiments, we show several robot configurations involving several types of modules. Each of these configurations matches the requirements of the task, and experiment 4, for instance, could only be solved using a configuration of two different robot module types. In fact, the LGP solver is very flexible and able to plan motions and actions of various robots without much adaption [31], [32], [36]. To plan new modular robot scenarios, programmers simply need to define the hardware constraints and action parameters in a *first-order-logic*-like language.

Another major challenge of modular robotic systems is the reconfiguration of the modules. There are systems that find optimal configurations but are unable to autonomously reconfigure the modules [16], [46], [47]. Other systems can self-reconfigure their modules, but fail to plan when and how it is ideal to reconfigure [11], [21], [48]–[50]. The work of Daudelin et al. [45] demonstrated that such planning is possible, but the systems capacities were limited. The integrated TAMP solver that we present, however, is able to not only realise that a task can only be executed if several modules connect, but also automatically find a model configuration and execute the reconfiguration fully autonomously. Different configurations were found, such as a mobile manipulator comprised of two modules and a crawler-like configuration of two modules. Our experiments demonstrate that the LGP-based heuristic TAMP planner for modular robots can perform autonomous reconfiguration planning and autonomously execute tasks that go beyond the capabilities of previous autonomous systems, that were restricted locomotion or exploration [11], [48]. Although the task in experiment 3 seems intuitively easy, it is quite challenging for TAMP solvers: there are many objects in the scene, there are multiple

agents that can execute actions at any time, and there are multiple goals. Nevertheless, the presented LGP planner could solve the task with up to 8 stairs in the scene. In addition, the RH-LGP solver could also solve it for 16 stairs. We are not aware of any previous works that have been able to plan and execute high-level tasks of modular robots necessitating a similar number of actions, including autonomous self-reconfiguration, as we are able to demonstrate in the present work.

5.3 Search space reduction through the use of heuristics

In addition to the demonstration of flexible TAMP in complex scenarios for modular robots, we present a problem independent heuristics interface for the LGP planner. Some existing heuristic planners are constructed to specifically function with a given heuristic. In comparison to some learning-based approaches [32], [67], which tested their task-specific heuristics only in a single type of scenario, e.g. pick-and-place, the mix-and-match principle of our heuristics make the heuristics applicable to a wide range of tasks. Consequently, we demonstrate with our experiments that pick-and-place, but also navigation and climbing tasks can be solved by the same planner. It was only required to exchange the specific action heuristics that we used to compute the cost-to-go values, along with minor adjustments, such as the initial cost-to-go value that we used in a specific scenario. Thus, a main strength of our interface is its problem independence.

In general, it is also possible to use the interface without any of the action-specific heuristics that we describe in the present paper. Yet, we also demonstrate in experiment 1, that these heuristics help to guide the symbolic search effectively. The standard MBTS approach [31] speeds up TAMP planning by reducing the complexity of most of the NLPs that are solved during planning. However, this approach performs standard search-based task planning without much consideration of the geometric properties of the environment. The problem-independent interface for heuristics that we introduce can be used to guide symbolic planning in LGP using geometric information. In doing so, we show that planning long-horizon tasks becomes computationally tractable. In experiment 1, we showed that problem instances which are 16 times bigger than the largest problem instance that was tractable for the classic LGP planner could be solved using the geometry-based heuristics. For all scenarios that could be solved by both planners, the heuristics-based planner expanded fewer nodes than the classic LGP solver using MBTS. In addition to the number of expanded nodes, the general size of the search tree was much smaller for the heuristics-aided planning approach, which had a smaller search tree when solving the scenario with 32 stairs than the standard planner when solving the problem with only 4 stairs. These results do not only validate the idea of leveraging geometric information during MBTS, but they also highlight the versatility and efficacy of our approach. Our approach also compares well to other heuristic-based LGP modifications: in [32], [37] the learned heuristics only help to find action sequences of six actions or less. Our planner was able to find action sequences of up to 43 actions in experiment 2 however.

5.4 A long-horizon planner

Previous works that used heuristics for TAMP mostly focused on smaller problem instances, which either required few actions to be solved [32], [71] or contained few objects in the environment [31]. Hartmann et al. however demonstrated that LGP is suitable for long-horizon problems [36]. To this day though, we are not aware of any TAMP algorithm for modular robots that solved complex tasks involving both self-reconfiguration and a great number of actions. In our experiments, the presented planner was able to plan such complex tasks: in the first experiment, we planned a task that required 43 actions, in the third experiment, several types of actions were necessary to complete multiple goals, and in the last experiment, we demonstrated that our planner can plan and execute sophisticated self-reconfiguration followed by a pick-and-place task. In addition, we introduced the RH-LGP solver that further boosted the performance of the heuristics-based LGP planner as we could show in experiment 5. Despite previous works using iterative TAMP approaches as well [36], [38], [72], [73], [75], we are not aware of any previous works that combined RHC with TAMP of modular robots or LGP. In the last experiment we were able to prove that the RH-LGP solver is able to solve long-horizon manipulation problems. In addition, for all but one scenario, the tractable problem size was increased by using the RH-LGP solver. It is very notable that experiment 5 also shows that tasks for modular and non-modular robots were handled by the same RH-LGP solver. This underlines the potential of the presented planner for any robotic manipulation task.

It may be considered a limitation of the RH-LGP planner that it cannot backtrack to previous LGP instances if it fails to find a solution for the next time horizon. However, it would be very inefficient if previously solved sub-problems would be kept in memory in order to enable backtracking. In fact, the planning performance is improved by this as old parts of the search space are removed from memory. Generally, the iterative LGP approach benefits from two aspects: the size of the symbolic search space is reduced by re-initialising the search tree upon each iteration, and sampling-based motion planning is more stable for shorter action sequences [36]. We believe that these properties make the RH-LGP planner suitable for many non-modular robot tasks as well.

5.5 Limitations

Despite the various contributions that we make, our work has several limitations that we discuss below.

The reconfiguration planning mechanism is not optimal

In comparison to other works focusing solely on the reconfiguring planning of modular robots, our method has limitations. So far, the only way that non-optimal *connect* and *disconnect* actions are pruned during search are if the two actions in immediate succession, or connections attempts of modules that are too distant. To incentivise reconfiguration, we discounted the heuristic costs of connect actions in the first cost-to-go function we used in the first two experiments. However, this incentive was only verified empirically, but lacks theoretical foundation. Some other works have indeed developed specific reconfiguration-planning algorithms, which optimise reconfiguration

planning [76], [77]. However, we were not able to incorporate these approaches into our planner due to the computational costs associated with solving another optimisation problem for a given action sequence. If we were able to ensure that the robot configurations in our TAMP solutions are optimal, we may be able to significantly reduce both costs and energy of the task execution. At the moment, this is only done for the motions that are executed during a feasible action sequence, but not on task planning level.

Integrating the individual heuristics in admissible manner is difficult

Previous works that used geometric knowledge as a heuristic for task planning did not provide optimality or admissibility guarantees of their heuristics [32], [68], [78]. From a theoretical perspective, we also struggle to combine the different action-specific heuristics in optimal manner. Despite the individual heuristics only prune futile actions sequences, we have not found a way to combine these heuristics in the interface function in a way that guarantees admissibility. In particular, defining appropriate initial cost-to-go values is difficult. If we were to integrate the individual action costs in an admissible way, our planning algorithm would inherit A*-like optimality.

The heuristic object position approximation is simple

The approximation of the robot position that we make during symbolic planning is based on the objects that the robots are standing on. This approximation makes the assumption that any object that the robot can walk on does not move. This assumption does not hold for all real life scenarios though. We believe that the introduction of the RH-LGP solver might alleviate this problem, as the kinematics of the environment are updated after each planning iteration. Nevertheless, it remains a limitation of our heuristic position approximation that we only consider the position of the base objects as a single point in space. This approximation neglects the actual shape of objects as it for example reduced a whole plane to a single point. Thus, it could occur that very large base objects overlap, but their positions in the sense how we defined are very far apart. In such case, stepping from one object onto the other would be avoided by the planner, although being feasible.

The RH-LGP solver fails to solve certain scenarios

A further limitation of the present work lies in the results of experiment 5. For 64 objects in the second scenario, the RH-LGP solver was not able to find a solution for no horizon length that we tried. We believe that this is due to the computational costs of symbolic search. Although only a limited amount of motions is planned and executed in each iteration, task planning still involves solving the entire remaining task. In the first planning iteration for instance, this corresponds to a similar task planning effort as when the task is planned with the standard LGP solver. Since 64 objects was the most that the heuristics-based LGP solver could solve in experiments 1 & 2, it is likely that RH-LGP failed to solve this problem because symbolic planning was too costly. This effect may have been intensified by the setup we used for our experiments (Intel i5-8250U CPU and 8GB RAM). A possible solution to this issue may be to accept symbolic solutions that

meet sub-goals instead of all overall task goals. So far, the RH-LGP planner that we present is trying to meet every symbolic goal in each planning iteration. An alternative approach could see the planner evaluate the symbolic state after each iteration and remove fulfilled sub-goals from the input goals. This way, the initial planning effort is reduced, but the total number of planning iterations may increase.

Another limitation of the RH-LGP solver is that we observed that for some combinations of horizon length and task, the solver can get stuck in unfortunate kinematic states and be unable to solve the task. This is due to the fact that the planner cannot backtrack once an iteration is finished. Thus, it cannot be assumed that if the planner can solve a task with horizon length n, it can also solve it for all other possible horizon lengths.

5.6 Future work

The largest robot configuration that we present in our work is the mobile manipulator, consisting of two robot modules. Future research should try to go beyond and investigate larger configurations. At the moment, it is not possible for more than two crawlers to form a chain, because the *stepTogether* decision rule has to invert every joint in the kinematic chain. So far, we only managed to hard-code this for every joint, as the graph language that we use does not have looping-capacities. Thus, we encourage research that tries to generalise the step and connect decision rules, such that it becomes possible to chain more than two crawlers.

In addition, other forms of connections between the robots could be explored. Throughout the project we started to explore the possibility of four crawlers to connect and perform quadrupedal locomotion. Unfortunately, it was not possible to realise this in the given amount of time, but it generally seems feasible and could be another great addition to the system.

Furthermore, the heuristics that we use can be improved. Apart from the step heuristic, the action-specific heuristics that we use are quite uninformed. For instance, the grasp heuristic only prunes geometrically infeasible grasps, but it does not inform the search in the sense that grasping actions are favoured at the right time during task planning. Due to the problem independence of the heuristics interface, it would be easy for future research projects to bring forward new heuristics that can be used to improve symbolic planning.

Lastly, we were not able to test the performance of the RH-LGP solver in a dynamic environment, where objects in the scene are moving without influence of a robot manipulator. This is unfortunately due to the limited time horizon of this project. Since we could test the RH-LGP planner in various static environments though, we believe that it would also be able to succeed in dynamic environments, if the horizon length is chosen appropriately. Hence, we encourage future research to deploy the RH-LGP solver in dynamic environments.

6 Conclusion

In this paper, we introduced a TAMP solver, incorporating geometry-based heuristic knowledge and enabling iterative planning, to solve long-horizon tasks for modular robots. Our work contributes with an important first step towards the development of fully autonomous modular robots that can solve a variety of complex and long-horizon tasks.

A main challenge for TAMP is the complexity of long-horizon tasks including having several robots in the environment. The proposed idea is to extend LGP with an interface for heuristic knowledge to speed up task planning, and secondly to limit the size of the considered problem by iteratively solving it. The heuristics that we use approximate the actual robot positions by using the position of the objects that the robots are standing on. The iterative TAMP solver, called RH-LGP, plans and executes a given number of actions in each step. The solver has been implemented and evaluated in simulation and our experimental results across five experiments show that combining heuristics and iterative planning renders long and difficult tasks tractable. This property makes the solver significantly more efficient than other TAMP solvers and enable planning tasks and motions of modular and non-modular robots.

List of Figures

List of Tables

References

[1] R. Bogue, "Domestic robots: Has their time finally come?" *Industrial Robot: An International Journal*, 2017.

[2] R. Works, "The impact of technology on labor markets," *Monthly Labor Review*, pp. 1–2, 2017.

[3] K. S. Saidi, T. Bock, and C. Georgoulas, "Robotics in construction," in *Springer handbook of robotics*, Springer, 2016, pp. 1493–1520.

[4] (Sep. 8, 2020). "Großauftrag: Kuka liefert 185 schweißroboter nach korea," [Online]. Available: `https://www.blechonline.de/grossauftrag-kuka-liefert-185-schweissroboter-nach-korea` (visited on 06/21/2021).

[5] (Aug. 25, 2010). "Packaging robots: Man's best friend," [Online]. Available: `https://www.packaging-gateway.com/features/94125/` (visited on 06/21/2021).

[6] A. M. Gillinov, T. Mihaljevic, H. Javadikasgari, R. M. Suri, S. L. Mick, J. L. Navia, M. Y. Desai, J. Bonatti, M. Khosravi, J. J. Idrees, *et al.*, "Early results of robotically assisted mitral valve surgery: Analysis of the first 1000 cases," *The Journal of thoracic and cardiovascular surgery*, vol. 155, no. 1, pp. 82–91, 2018.

[7] E. Oña, R. Cano-de La Cuerda, P. Sánchez-Herrera, C. Balaguer, and A. Jardón, "A review of robotics in neurorehabilitation: Towards an automated process for upper limb," *Journal of healthcare engineering*, vol. 2018, 2018.

[8] N. Tan, A. A. Hayat, M. R. Elara, and K. L. Wood, "A framework for taxonomy and evaluation of self-reconfigurable robotic systems," *IEEE Access*, vol. 8, pp. 13 969–13 986, 2020.

[9] S. Robotics. "Meet thorvald - the modular agricultural multipurpose robot," [Online]. Available: `https://sagarobotics.com/pages/thorvald-platform` (visited on 07/15/2021).

[10] F. Mondada, M. Bonani, S. Magnenat, A. Guignard, D. Floreano, F. Groen, N. Amato, A. Bonari, E. Yoshida, and B. Kröse, "Physical connections and cooperation in swarm robotics," in *8th Conference on Intelligent Autonomous Systems (IAS8)*, 2004, pp. 53–60.

[11] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: A modular reconfigurable robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 1, 2000, pp. 514–520.

[12] N. Melenbrink, P. Kassabian, A. Menges, and J. Werfel, "Towards force-aware robot collectives for on-site construction," 2017.

[13] M. Ning, L. Shao, F. Chen, M. Li, C. Zhang, and Q. Zhang, "Modeling and analysis of a modular multilegged robot with improved fault tolerance and environmental adaptability," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[14] A. Brunete, A. Ranganath, S. Segovia, J. P. de Frutos, M. Hernando, and E. Gambao, "Current trends in reconfigurable modular robots design," *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, p. 1 729 881 417 710 457, 2017.

[15] J. Liu, X. Zhang, and G. Hao, "Survey on research and development of reconfigurable modular robots," *Advances in Mechanical Engineering*, vol. 8, no. 8, p. 1 687 814 016 659 597, 2016.

[16] M. Althoff, A. Giusti, S. B. Liu, and A. Pereira, "Effortless creation of safe robots from modules through self-programming and self-verification," *Science Robotics*, vol. 4, no. 31, 2019.

[17] V. Prabakaran, M. R. Elara, T. Pathmakumar, and S. Nansai, "Htetro: A tetris inspired shape shifting floor cleaning robot," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 6105–6112.

[18] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[19] G. Liang, H. Luo, M. Li, H. Qian, and T. L. Lam, "Freebot: A freeform modular self-reconfigurable robot with arbitrary connection point-design and implementation," in *IEEE/RSJ Int. Conf. Intell. Robots Syst., Las Vegas, USA*, 2020.

[20] N. Melenbrink, J. Werfel, and A. Menges, "On-site autonomous construction robots: Towards unsupervised building," *Automation in Construction*, vol. 119, p. 103 312, 2020.

[21] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[22] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[24] E. Karpas and D. Magazzeni, "Automated planning for robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 417–439, 2020.

[25] S. H. Tang, W. Khaksar, N. Ismail, and M. Ariffin, "A review on robot motion planning approaches," *Pertanika Journal of Science and Technology*, vol. 20, no. 1, pp. 15–29, 2012.

[26] S.-T. Nguyen, O. S. Oguz, V. N. Hartmann, and M. Toussaint, "Self-supervised learning of scene-graph representations for robotic sequential manipulation planning,"

[27] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning.," in *IJCAI*, 2015, pp. 1930–1936.

[28] H. M. Van, O. S. Oguz, Z. Zhou, and M. Toussaint, "Guided sequential manipulation planning using a hierarchical policy," in *RSS Workshop on Learning in Task and Motion Planning*, 2020.

[29] A. Akbari, J. Rosell, *et al.*, "Task planning using physics-based heuristics on manipulation actions," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2016, pp. 1–8.

[30] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[31] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 4044–4051.

[32] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," *arXiv preprint arXiv:2006.05398*, 2020.

[33] M. Bordignon, K. Stoy, and U. P. Schultz, "A virtual machine-based approach for fast and flexible reprogramming of modular robots," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 4273–4280.

[34] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez, "A long horizon planning framework for manipulating rigid pointcloud objects," *arXiv preprint arXiv:2011.08177*, 2020.

[35] Z. Zhao, Z. Zhou, M. Park, and Y. Zhao, "Sydebo: Symbolic-decision-embedded bilevel optimization for long-horizon manipulation in dynamic environments," *arXiv preprint arXiv:2010.11078*, 2020.

[36] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges, "Robust task and motion planning for long-horizon architectural construction planning," *arXiv preprint arXiv:2003.07754*, 2020.

[37] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint, "Learning geometric reasoning and control for long-horizon tasks from visual input," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[38] N. Castaman, E. Pagello, E. Menegatti, and A. Pretto, "Receding horizon task and motion planning in dynamic environments," *arXiv preprint arXiv:2009.03139*, 2020.

[39] J. Mattingley, Y. Wang, and S. Boyd, "Receding horizon control," *IEEE Control Systems Magazine*, vol. 31, no. 3, pp. 52–65, 2011.

[40] H. Kawano, "Linear heterogeneous reconfiguration of cubic modular robots via simultaneous tunneling and permutation," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 332–338.

[41] Z. H. Ismail and N. Sariff, "A survey and analysis of cooperative multi-agent robot systems: Challenges and directions," in *Applications of Mobile Robots*, IntechOpen, 2018, pp. 8–14.

[42] A. Lyder, K. Stoy, R. Garciá, J. C. Larsen, and P. Hermansen, "On sub-modularization and morphological heterogeneity in modular robotics," in *Intelligent Autonomous Systems 12*, Springer, 2013, pp. 649–661.

[43] S. Leder, R. Weber, D. Wood, O. Bucklin, and A. Menges, "Design and prototyping of a single axis, building material integrated, distributed robotic assembly system," in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2019, pp. 211–212. DOI: 10.1109/FAS-W.2019.00056.

[44] A. Brunete, M. Hernando, E. Gambao, and J. E. Torres, "A behaviour-based control architecture for heterogeneous modular, multi-configurable, chained micro-robots," *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1607–1624, 2012.

[45] J. Daudelin, G. Jing, T. Tosun, M. Yim, H. Kress-Gazit, and M. Campbell, "An integrated system for perception-driven autonomy with modular robots," *Science Robotics*, vol. 3, no. 23, 2018.

[46] M. Gauci, R. Nagpal, and M. Rubenstein, "Programmable self-disassembly for shape formation in large-scale robot collectives," in *Distributed Autonomous Robotic Systems*, Springer, 2018, pp. 573–586.

[47] T. K. Tucci, B. Piranda, and J. Bourgeois, "A distributed self-assembly planning algorithm for modular robots," in *International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[48] B. Salemi, M. Moll, and W.-M. Shen, "Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2006, pp. 3636–3641.

[49] V. Zykov, A. Chan, and H. Lipson, "Molecubes: An open-source modular robotics kit," in *IROS-2007 Self-Reconfigurable Robotics Workshop*, Citeseer, 2007, pp. 3–6.

[50] D. Rus and M. Vona, "A physical implementation of the self-reconfiguring crystalline robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 2, 2000, pp. 1726–1733.

[51] C. Liu, S. Yu, and M. Yim, "Motion planning for variable topology truss modular robot," in *Proceedings of Robotics: Science and Systems*, 2020.

[52] C. Liu, M. Whitzer, and M. Yim, "A distributed reconfiguration planning algorithm for modular robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4231–4238, 2019.

[53] T. Lozano-Perez, "Spatial planning: A configuration space approach," in *Autonomous robot vehicles*, Springer, 1990, pp. 259–271.

[54] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 5002–5008.

[55] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 7087–7094.

[56] S. Srivastava, L. Riano, S. Russell, and P. Abbeel, "Using classical planners for tasks with continuous operators in robotics," in *Intl. Conf. on Automated Planning and Scheduling*, Citeseer, vol. 3, 2013.

[57] J. O. de Haro, V. N. Hartmann, O. S. Oguz, and M. Toussaint, "Learning efficient constraint graph sampling for robotic sequential manipulation," *arXiv preprint arXiv:2011.04828*, 2020.

[58] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Scalable lazy smt-based motion planning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, IEEE, 2016, pp. 6683–6688.

[59] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 3684–3691.

[60] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*, Springer, 2015, pp. 179–195.

[61] 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 *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 639–646.

[62] J. Ferrer-Mestres, G. Frances, and H. Geffner, "Combined task and motion planning as classical ai planning," *arXiv preprint arXiv:1706.06927*, 2017.

[63] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[64] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Ffrob: Leveraging symbolic planning for efficient task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[65] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 04, pp. 479–503, 2005.

[66] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[67] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 447–454.

[68] S. W. Yoon, A. Fern, and R. Givan, "Learning heuristic functions from relaxed plans.," in *ICAPS*, vol. 2, 2006, p. 3.

[69] D. Driess, O. Oguz, and M. Toussaint, "Hierarchical task and motion planning using logic-geometric programming (hlgp)," in *RSS Workshop on Robust Task and Motion Planning*, 2019.

[70] W. Vega-Brown and N. Roy, "Admissible abstractions for near-optimal task and motion planning," *arXiv preprint arXiv:1806.00805*, 2018.

[71] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[72] S. Nair and C. Finn, "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation," *arXiv preprint arXiv:1909.05829*, 2019.

[73] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," *arXiv preprint arXiv:1910.11956*, 2019.

[74] M. Toussaint, "Newton methods for k-order markov constrained motion problems," *arXiv preprint arXiv:1407.0414*, 2014.

[75] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," *arXiv preprint arXiv:2012.07277*, 2020.

[76] I.-M. Chen and J. W. Burdick, "Determining task optimal modular robot assembly configurations," in *proceedings of 1995 IEEE International Conference on Robotics and Automation*, IEEE, vol. 1, 1995, pp. 132–137.

[77] M. Yao, X. Xiao, C. H. Belke, H. Cui, and J. Paik, "Optimal distribution of active modules in reconfiguration planning of modular robots," *Journal of Mechanisms and Robotics*, vol. 11, no. 1, p. 011 017, 2019.

[78] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

Appendix

Decision rules for modular robots

Logical predicates are displayed in round brackets, i.e. (gripper A). Motion constraint predicates are displayed in square brackets, i.e., [stable X Y].

`(step A B X Y):`

PRE: (gripper A) (support A) (busy A) (gripper B) ¬(support B) ¬(busy B) (linked A B) ¬(connected A B) (object X) (base X) (on X A) (object Y) (base Y) ¬(grippable X) ¬(grippable Y) ¬(INFEASIBLE step A B X Y)

POST: ¬(busy A) ¬(support A) ¬[stable ANY A] ¬(on X A) ¬[touch A X] (busy B) (support B) (on Y B) [stable Y B] [touch Y B]

`(grasp A X):`

PRE: (gripper A) ¬(busy A) ¬(support A) (object X) ¬(gripper X) (grippable X) ¬(held X) ¬(placed X) ¬(INFEASIBLE grasp A X)

POST: ¬[stable ANY X] (busy A) (held X) (grasped A X) [stable A X] [touch A X]

`(place A X Y):`

PRE: (gripper A) (grasped A X) (tray Y) (held X) ¬(INFEASIBLE place A X Y)

POST: ¬(grasped A X) ¬(busy A) ¬(busy X) ¬(held X) (placed X) ¬[stable X ANY] (on Y X) ¬[touch A X] [touch X Y] [above X Y] [stable Y X] ¬[stable A X]

`(connect A B C D):`

PRE: (gripper A) (support A) (busy A) ¬(onMobile A) (gripper B) ¬(support B) ¬(busy B) (linked A B) (gripper C) ¬(support C) ¬(busy C) (gripper D) (support D) (busy D) (linked C D) ¬(equal B C) ¬(equal A D) ¬(INFEASIBLE connect A B C D)

POST: ¬(busy A) ¬(support A) ¬[stable ANY A] ¬(on ANY A) (busy B) (busy C) (connected B C) (connected C B) [stable C B] [connectCrawlers C B]

`(disconnect A B C D X):`

PRE: (gripper A) ¬(support A) ¬(busy A) (busy B) (busy C) (linked A B) ¬(equal A B) (linked C D) ¬(equal C D) (connected B C) (gripper D) (support D) (on X D) ¬(INFEASIBLE disconnect A B C D X)

POST: (support A) (busy A) [stable X A] [touch X A] (on X A) ¬(busy B) ¬(support B) ¬[stable ANY B] ¬(busy C) ¬(support C) ¬[stable C ANY] ¬[stable B C] ¬(connected B C) ¬(connected C B) ¬[connectCrawlers B C] ¬[connectCrawlers C B] ¬[touch B C] ¬[touch B C]

`(stepTogether A B C D X Y):`

PRE: (gripper A) (support A) (busy A) (gripper B) ¬(support B) (busy B) (gripper C) ¬(support C) (busy C) (gripper D) ¬(support D) ¬(busy D) (connected B C) (linked A B) (linked C D) (object X) ¬(grippable X) (object Y) ¬(grippable Y) (on X A) ¬(INFEASIBLE stepTogether A B C D X Y)

POST: ¬(busy A) ¬(support A) ¬[stable ANY A] ¬(on X A) (busy D) (support D) [stable Y D] [touch Y D] (on Y D) ¬[stable B ANY] ¬[stable ANY C] [touch C B] [stable C B]

`(connect2mobileBase A B X Y):`

PRE: (gripper A) (support A) (busy A) (gripper B) ¬(support B) ¬[stable ANY B] ¬(busy B) (linked A B) ¬(connected A B) (object X) ¬(gripper X) (on X A) (object Y) (mobile Y) ¬(busy Y) ¬(INFEASIBLE connect2mobileBase A B X Y)

POST: ¬(busy A) ¬(support A) ¬[stable ANY A] ¬(on X A) (busy B) (support B) [stable Y B] (on Y B) (busy Y) (onMobile B) [touchBoxNormalX B Y]

Action specific heuristics

Action	Heuristic

grasp	**Function** `graspHeuristic`(*gripper* g, *object* o)**:**
	  if *reachable*(g, o) **then**
	  return 0;
	  else
	  return 1000;

place	**Function** `placeHeuristic`(*gripper* g, *object* o, *target* t)**:**
	  if *reachable*(g, t) $\wedge$ t == *goalTray* **then**
	  return 0;
	  else
	  return 1000;

step / stepTogether	**Function** `stepHeuristic`(*currentBase* c, *stepTargetBase* t, *navigationGoal* g, *moduleCount* n)**:**
	  nearby $\leftarrow$ $\{o \mid$ object(o) $\wedge$ dist(c, o) $<$ crawlerLen[n]$\}$;
	  if *nearby.contains*(t) **then**
	  stepSize $\leftarrow$ dist(t,g) - dist(c,g);
	  return stepSize;
	  else
	  return 1000;

connect	**Function** `connectHeuristic`(*priorDecision* d, *effectorA* a, *effectorB*, b)**:**
	  if *reachable*(a,b) $\wedge$ d $\neq$ *disconnect* **then**
	  return 0;
	  else
	  return 1000;

disconnect	**Function** `disconnectHeuristic`(*priorDecision* d)**:**
	  if d $\neq$ *connect* **then**
	  return 0;
	  else
	  return 1000;

Table 8: Action specific heuristics used in the present work.

The cost-to-go functions that were passed to the interface in the experiments

Algorithm 2: Cost-to-go used for experiments 1 and 2.

```
1  Function costToGo1&2(SearchTreeNode n, target t):
2      if n.level == 0 then
3          n.costToGo ← min_{robots c} dist(c,t);
4          return;
5      n.costToGo ← n.parent.costToGo;
6      action ← n.sequence.last();
7      if action is "step" then
8          n.costToGo += stepHeuristic(action.from, action.to, t, 1);
9      else if decision is "stepTogether" then
10         n.costToGo += stepHeuristic(action.from, action.to, t, 2);
11     else if decision is "connect" then
12         n.costToGo += connectHeuristic(action.parent, action.effector1,
             action.effector2) - 1.5;
13     else if decision is "disconnect" then
14         n.costToGo += disconnectHeuristic(action.parent);
15     else if decision is "grasp" then
16         n.costToGo += graspHeuristic(action.effector1, action.object);
```

Algorithm 3: Cost-to-go used for experiment 3.

```
1  Function costToGo3(SearchTreeNode n, targets):
2      if n.level == 0 then
3          n.costToGo ← dist(crawlerA, crawlerB) + min_{robots c} dist(c,target);
4          return;
5      n.costToGo ← n.parent.costToGo;
6      action ← n.sequence.last();
7      if action is "step" then
8          otherCrawler ← n.environment.otherRobot;
9          n.costToGo += stepHeuristic(action.from, action.from,
             otherCrawler.position, 1);
10     else if decision is "stepTogether" then
11         n.costToGo += min_{targets t} stepHeuristic(action.from, action.to, t, 2);
12     else if decision is "connect" then
13         n.costToGo += connectHeuristic(action.parent, action.effectorA,
             action.effectorB);
14     else if decision is "disconnect" then
15         n.costToGo += disconnectHeuristic(action.parent);
16     else if decision is "grasp" then
17         n.costToGo += graspHeuristic(action.effectorA, action.object);
```

Algorithm 4: Cost-to-go function used for experiment 4.

```
1  Function costToGo4(SearchTreeNode n, targetTray t):
2      if n.level == 0 then
3          n.costToGo ← number of target objects;
4          return;
5      n.costToGo ← n.parent.costToGo;
6      action ← n.sequence.last();
7      if action is "connect2mobile" then
8          n.costToGo += -0.15;
9      else if decision is "connect" then
10         n.costToGo += connectHeuristic(action.parent, action.effectorA,
           action.effectorB);
11     else if decision is "disconnect" then
12         n.costToGo += disconnectHeuristic(action.parent);
13     else if decision is "grasp" then
14         n.costToGo += graspHeuristic(action.effectorA, action.object);
```

Department I – Student Services
Examinations

Technische Universität Berlin

Guidance on Completing *Diplom*, Bachelor's and Master's Theses

The following guidance is intended to make it easier for you to complete your final theses.

Please be aware of the following guidelines:

1. The writing period begins on the day the topics are distributed, either in person or via post. This will, in general, be done by Examinations. The writing period ends on the date specified.

2. Applications for extensions are to be submitted to Examinations **before the writing period has expired.**

   In general, when applying for an extension due to illness, it is sufficient to submit a medical certificate from a doctor with the application for an extension to the writing period. A certificate of incapacity is not required.

   An application for an extension arising from issues related to content must include a statement from the referee.

3. **In general, the examination regulations of your own degree program apply.** This may mean that your thesis will need to include a short summary in German and/or English.

   If a thesis is **not** in German, the approval of the referee must be provided when submitting the application for admission to the thesis. In such cases a short German-language summary must also be included when submitting the thesis itself.

   Some examination regulations also require that a thesis be additionally submitted in electronic format.

4. For group work projects, individual contributions must be clearly identifiable. It is to be made absolutely clear which sections of a thesis were completed by which student (for example by means of sections, page numbers or other transparent criteria). It must be clearly specified in the application for admission that the thesis is to be completed as a group project.

5. Quotations and text contributions from other works (e.g. specialist literature, commentaries, other theses or the Internet) whether quoted literally or paraphrased, must be acknowledged clearly in the references. Longer passages of direct quotation should be clearly identified as such either by indenting, italicizing, setting in brackets or by other suitable means. Failure to use such methods to indicate quotations could be seen as an attempt to plagiarize, which will, should the suspicion of plagiarism be confirmed, in turn lead to a thesis not achieving a pass grade. If the number of attempts permitted to pass would not thus be exceeded, it is possible to reapply to write and submit a thesis, though with a different topic.

6. **Three copies** of a thesis are to be submitted to Examinations. The copies are to be bound, i.e. laminated, glued or spiral-bound; ring binders are not acceptable. This does not apply to plans, models and similar material, which should be separately submitted along with the thesis.

IB / Stand: Oktober 2017

Technische Universität Berlin

Final theses can only be accepted by Examinations after at least half the writing time allotted by the relevant subject-specific Study and Examinations Regulations has expired. Any final thesis submitted to Examinations prior to the expiry of half the allotted writing time, must include a statement from the first referee, endorsed by the Examination Board, justifying the reason for the early submission.

Should the submission deadline for the thesis fall on a Sunday or a public holiday, a Saturday or a day on which Examinations is not open, the deadline is extended to the following workday or the next day that Examinations is open. No application for an extension is then required.

This regulation does not apply to discontinuing *Diplom* degrees, for which the submission date is binding.

Theses can also be handed in to the main reception or in the Campus Center. Please note that receipt must be documented.

7. A sworn affidavit, serving as the first page of the thesis, is to be included, bearing the following statement:

   I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

   Place, Date

   Berlin, 8.8.21

   .....

   Signature

8. For any minor editorial changes or digressions from the agreed topic, the written approval of the referee must be provided, at the latest when submitting the thesis.

We very much hope this information will prove helpful to you with all formal issues associated with writing your thesis and we wish you every success.

Your Examinations team.

Technische Universität Berlin

Guidance for the Submission of *Diplom*/Bachelor's/Master's Theses (Application for admission)

1. The *Diplom*/ Bachelor's/Master's thesis is to be submitted to Examinations – IB 2- Room 0010 or in the Campus Center **at the latest on the last day of the writing period** (see application for admission).

2. **Opening Times:**
   Monday, Thursday and Friday: 9.30 – 12.30,
   Tuesday: 13.00-16.00,
   Wednesday: closed.
   (Should the submission deadline fall on a day where there are no office hours, submission is possible during the opening hours of the following work day).

3. The following number of (bound) ***copies** are to be submitted:

Degree programs:		**always german & english abstract included**
Biomedical Engineering (Master)	3	–
Computer Science (Bachelor)	3	
Computer Science (Master)	3	for PO 2013
Global Production Engineering (Master)	3	
Information Systems Management (Bachelor)	3	for PO 2013
Information Systems Management (Master)	3	for PO 2013
Physics (*Diplom*)	2	
Physics (Bachelor/Master)	3	
Production Engineering (Master)	3	
Transport Systems (Bachelor)	3	

4. **The topic** of the thesis must match verbatim with the topic on the application for admission. (**Any differences** require prior written confirmation from the examiner that they will not result in any changes to the content of the *Diplom*, Bachelor's or Master's thesis).

5. The following declaration is to be printed and signed and bound with the submitted thesis:

   I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

   The independent and unaided completion of the thesis is affirmed by affidavit:

   Berlin, C. [signature], 8.8.21 ***(Date)***

   Signature

6. Sources used have to be acknowledged as well as ideas borrowed from other theses.

7. **Group work:**
   The contributions of the individual authors have to be clearly identifiable (using sections, page numbers and other objective criteria which make a clear distinction possible).