Efficient Path Planning for Modular Reconfigurable Robots

Matthias Mayer, Zihao Li, and Matthias Althoff

***Abstract*— Industrial robots are essential for modern production but often struggle to adapt to new tasks. Modular (reconfigurable) robots can overcome this challenge by eliminating the need to replace the whole robot. However, finding the optimal assembly for a task remains difficult because a valid path has to be computed for each generated assembly – consuming a significant fraction of the computation time. Similar to online path planning, where previous approaches adapt known paths to a changing environment, we show that transferring paths from previously considered module assemblies accelerates path planning for the next assemblies. On average, our method reduces the planning time for single-goal tasks by $50\,\%$. The usefulness of our method is evaluated by integrating it in a genetic algorithm (GA) for optimizing assemblies and evaluating it on our benchmark suite CoBRA. Within the optimization loop for modular robots, the time used to check a single assembly is shortened by up to $50\,\%$.**

I. Introduction

Modular (reconfigurable) robots realize more efficient automation, especially in small-scale manufacturing: They exploit economies of scale in robot production, adapt to changing applications, reduce the complexity of many tasks, and simplify maintenance [1]. Multiple companies and start-ups exploit these advantages, e.g., Beckhoff[1], Hebi[2], and RobCo[3].

Adapting modular robots to a specific task is a complex, hybrid optimization problem, requiring a joint optimization of which modules to assemble and how to move them to fulfill a given task [2]. Current optimization methods involve path planning for each feasible assembly, e.g., [3]–[9], which often dominates the computation time. To address this issue, we introduce *planning with reuse*, i.e., we adapt previously obtained paths to similar modular robot assemblies to reduce computation time, as sketched in Fig. 1.

Our novel approach involves retrieving and adapting a previous path to fulfill all user-provided specifications. We first demonstrate the efficacy of our method in tasks with a single goal, using a fixed set of known paths and assemblies. Subsequently, we extend our approach to tasks with multiple goals and integrate it into modular robot optimization based on genetic algorithms. Our experiments show a reduction in planning time of up to $50\,\%$.

All authors are with the Technical University of Munich, School of Computation, Information and Technology, Assoc. Professorship for Cyber Pphysical Systems, Boltzmannstraße 3, 85748, Garching, Germany. {matthias.mayer, zihao.li, althoff}@tum.de

[1]https://www.beckhoff.com/de-de/produkte/motion/atro-automation-technology-for-robotics/, accessed on July 4th, 2024.

[2]https://www.hebirobotics.com/, accessed on July 4th, 2024.

[3]https://www.robco.de, accessed on July 4th, 2024.

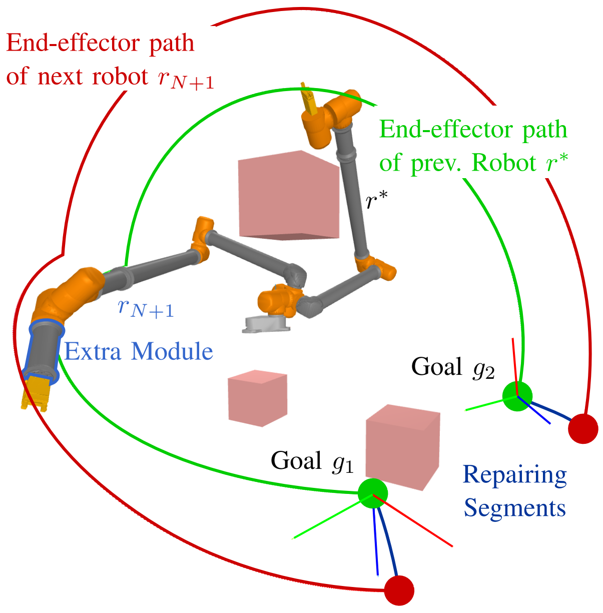

End-effector path of next robot r_{N+1}
End-effector path of prev. Robot r^*
r^*
r_{N+1}
Extra Module
Goal g_2
Goal g_1
Repairing Segments

Fig. 1. The green end-effector path p^* between goals g_1 and g_2 was planned for and executed on the previous assembly r^*. We modify the previous solution by reusing the old path of joint angles p^*, resulting in the red end-effector path of r_{N+1}, and appending the blue path segments.

A. Related Work

Path planning for robots has been studied for a long time [10]. One of the most commonly used approaches is sampling-based planning, which includes multi-query and single-query algorithms. The probabilistic roadmap planner (PRM) [11] and its variants are notable multi-query algorithms. A Rapidly-exploring random tree (RRT) [12] is a single-query algorithm with many variants growing a tree from the start configuration by randomly sampling the configuration space. In this work, we use RRT-Connect [13], which grows trees from the start and goal configurations until they connect; if RRT-Connect is used unaltered to plan a complete path, we call this *planning from scratch*.

While the initial idea of RRTs is to explore the configuration space, in practice, specific end-effector poses are usually desired, e.g., the pick-up location of an object, which requires the integration of inverse kinematics. The growth of rapidly-exploring random trees can be biased toward the desired goal pose [14] instead of towards a random direction. The work in [15] suggested a two-staged extension of RRTs, combining inverse kinematics with RRT-Connect.

Extending sampling-based planning to online methods that react to changing environments has been tackled, e.g., by the Lightening framework [16]. The authors propose a *retrieve-*

repair method that looks up previous planning results based on similar environments and repairs these retrieved paths such that they are collision-free in the new environment. Data from previous planning attempts can be stored, e.g., as sparse roadmaps (SPARS) [17] used by [16] or as found paths with their corresponding task [18]. Relevant previous planning attempts can be determined by minimizing a similarity function [19], [20].

Modular robots and their adaptation to specific tasks have been studied extensively; a small overview is given in [21, Ch. 22.2.5]. Proposed solutions to find optimal assemblies include the reduction of the search space to common robot kinematics [4], hierarchical testing with increasing complexity [3]–[5], or reducing the constraints on the solution by only requiring the existence of inverse kinematic solutions instead of complete trajectories [6].

B. Problem Statement

Following [2], we want to find modular robots assembled from a set of robot modules $\mathcal{R}$ that fulfill a task Θ. Each task specifies the constraint set $\mathcal{C}$ and a set of goals $\mathcal{G}$. Our goal is to find

- assembled robots r_i stored as a tuple of modules $r_i = (m_1, \dots, m_a), m_i \in \mathcal{R}$, with $n_i \in \mathbb{N}_{>0}$ degrees of freedom, and
- associated n_i-dimensional paths $p_i \colon [0,1] \to \mathbb{R}^{n_i}$ subject to the constraints $p_i(0) = \mathbf{q}_{\text{start}}, p_i(1) = \mathbf{q}_{\text{goal}}, \forall t \in [0,1], \forall c \in \mathcal{C} \colon c(p_i(t)) \leq 0$.

In contrast to [2], we only consider tasks Θ with a fixed robot base pose.

To leverage previous experience, we store all previously obtained valid paths $p_i, i \in 1, \dots, N, N \in \mathbb{N}$ for the corresponding robot assemblies r_i in a database $\mathcal{P} = \{(r_1, p_1), \dots, (r_N, p_N)\}$, where in contrast to [18], we store robots and not environment states alongside the paths. Promising previous paths $p^* \in \mathcal{P}$[4] are retrieved by our novel distance function and are adapted to the next assembly r_{N+1} by planning repair paths, as in [16].

C. Contributions

For the first time, we leverage previous experience in modular robot optimization to accelerate the underlying path planning. In contrast to similar works on online path planning, we consider a problem where the robot changes. In particular, we

- adapt path retrieval and repair to handle changing modular robot assemblies;
- reduce planning times for single and multiple goals;
- integrate our path planner into our modular robot optimization based on genetic algorithms.

In the next section, we present our method, including path retrieval in Sec. II-B, path repair in Sec. II-C, and integration into modular robot optimization in Sec. II-D. We present our numerical experiments in Sec. III, which we discuss in Sec. IV. Lastly, we summarize our results in Sec. V.

[4]We use this slight abuse of notation to indicate that p^* is an element in one of the tuples inside $\mathcal{P}$.

II. Method

Our new method *planning with reuse* is based on RRT-Connect and consists of two steps: path retrieval and path repair. A high-level overview of its sub-steps is provided in Fig. 2. First, given a database $\mathcal{P}$ of N previous assemblies and paths, *path retrieval* selects the most promising candidate $p^* \in \mathcal{P}$ (green in Fig. 1, 2a) from the database for reuse on the next assembly r_{N+1} (red in Fig. 1, 2). Second, *path repair* connects p^* to the goals in the workspace of r_{N+1}, using RRT-Connect to plan the connecting path segments (blue in Fig. 1, 2f). Our database $\mathcal{P}$ grows with each successfully evaluated assembly r_{N+1}, storing it with the found collision-free path p_{N+1}.

A. Notation

For each path p_i solving task Θ on assembly r_i, the configurations at the start and end of the path are denoted as $\mathbf{q}^s_{pi} \in \mathbb{R}^{n_i}$ and $\mathbf{q}^g_{pi} \in \mathbb{R}^{n_i}$; the corresponding end effector poses are $\mathbf{T}^{\text{start}}_{pi} \in \mathbb{SE}(3)$ and $\mathbf{T}^{\text{goal}}_{pi} \in \mathbb{SE}(3)$, which are calculated via the forward kinematics of assembly r_i [22]. Our method section will mostly focus on tasks with a single point-to-point (PTP) movement, defined by a desired start pose $\mathbf{T}_{\text{init}} \in \mathbb{SE}(3)$ and goal pose $\mathbf{T}_{\text{desired}} \in \mathbb{SE}(3)$. The configurations fulfilling $\mathbf{T}_{\text{init}}$ and $\mathbf{T}_{\text{desired}}$ with r_i are denoted as $\mathbf{q}^{ts}_i \in \mathbb{R}^{n_i}$ and $\mathbf{q}^{tg}_i \in \mathbb{R}^{n_i}$, and are calculated by inverse kinematics implemented in [22].

B. Path Retrieval

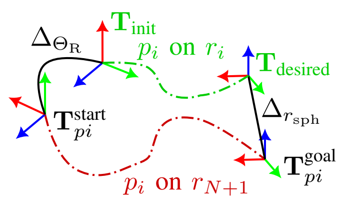
Path retrieval is implemented according to Alg. 1 and is used to find the best previous path $p^* \in \mathcal{P}$ to adapt to the next module assembly r_{N+1}. We implement this retrieval by minimizing a distance function, considering pose distance δ_1 and configuration distance δ_2. Figs. 2a-2e show the steps to find p^*. First, we select $0 < K \leq N$ candidates by minimizing the pose distance δ_1, then use the configuration distance δ_2 to obtain the best path p^*. Both distance criteria are based on the assumption that smaller joint space distances result in quicker repairs, which is validated in our numerical experiments. Next, the two distance functions are introduced.

1) Pose Distance δ_1: Our pose distance function takes the new assembly r_{N+1} and stored paths $p_i \in \mathcal{P}$ as inputs. Following [2], we use the shorthand $\Delta_S(\mathbf{T}, \mathbf{T}_{\text{desired}}) = S(\mathbf{T}^{-1}_{\text{desired}}\mathbf{T})$, to denote the distance between two poses under a projection S. Our work uses the Euclidean distance r_{sph} and the rotation angle Θ_{R} between the two poses to define

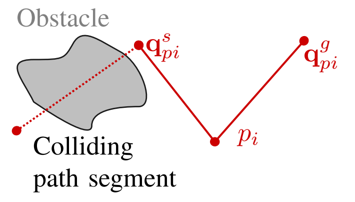
$$\begin{aligned}\delta_{1,i} =& \Delta_{r_{\text{sph}}}(\mathbf{T}^{\text{start}}_{pi}, \mathbf{T}_{\text{init}}) + \Delta_{\Theta_{\text{R}}}(\mathbf{T}^{\text{start}}_{pi}, \mathbf{T}_{\text{init}}) + \\ & \Delta_{r_{\text{sph}}}(\mathbf{T}^{\text{goal}}_{pi}, \mathbf{T}_{\text{desired}}) + \Delta_{\Theta_{\text{R}}}(\mathbf{T}^{\text{goal}}_{pi}, \mathbf{T}_{\text{desired}}),\end{aligned} \quad (1)$$

which is calculated for each $p_i \in \mathcal{P}$ by lines 1-2 in Alg. 1. The K path candidates and corresponding assemblies with the lowest distance δ_1 are selected in line 3.

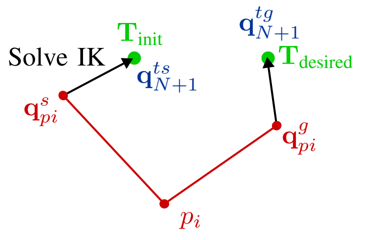
Intuitively, this distance helps to select paths that start and end close to the intended goals if executed on the new assembly r_{N+1}. In Fig. 2a, these distances are measured at either end of the path p_i executed on the old r_i (in green) and new assembly r_{N+1} (in red). We show the distance


$\mathbf{T}_{\text{init}}$
$\Delta_{\Theta_{\text{R}}}$
p_i on r_i
$\mathbf{T}_{\text{desired}}$
$\Delta_{r_{\text{sph}}}$
$\mathbf{T}_{pi}^{\text{start}}$
$\mathbf{T}_{pi}^{\text{goal}}$
p_i on r_{N+1}

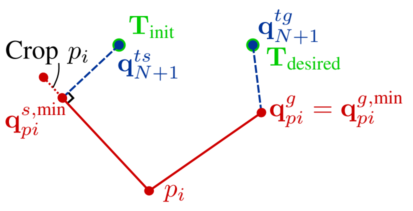
(a) Calculate $\delta_{1,i}$ in the workspace of the robot. Select K paths with lowest δ_1.



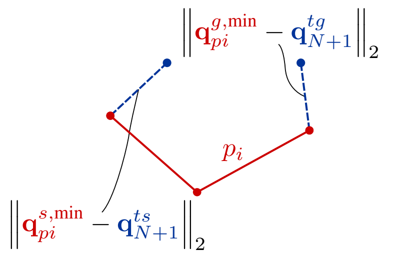
(b) Remove colliding path segments from p_i to get a valid section between $\mathbf{q}_{pi}^{s}$ and $\mathbf{q}_{pi}^{g}$.



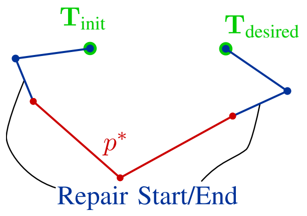
(c) Find inv. kinematic solutions $\mathbf{q}_{N+1}^{ts}, \mathbf{q}_{N+1}^{tg}$ close to $\mathbf{q}_{pi}^{s}, \mathbf{q}_{pi}^{g}$ (start/end of p_i).


$\mathbf{T}_{\text{init}}$
$\mathbf{q}_{N+1}^{tg}$
Crop p_i
$\mathbf{q}_{N+1}^{ts}$
$\mathbf{T}_{\text{desired}}$
$\mathbf{q}_{pi}^{s,\text{min}}$
$\mathbf{q}_{pi}^{g} = \mathbf{q}_{pi}^{g,\text{min}}$
p_i

(d) Min. distances from $\mathbf{q}_{N+1}^{ts}, \mathbf{q}_{N+1}^{tg}$ to p_i.


$\left\|\mathbf{q}_{pi}^{g,\text{min}} - \mathbf{q}_{N+1}^{tg}\right\|_2$
p_i
$\left\|\mathbf{q}_{pi}^{s,\text{min}} - \mathbf{q}_{N+1}^{ts}\right\|_2$

(e) Calculate δ_2; select minimum as p^*.



(f) Plan repairs for p^* resulting in p_{N+1}.

Fig. 2. We show the sub-steps of path planning with reuse, mostly in joint space. 2a - 2e retrieve the candidate path p^* to adapt to a new assembly r_{N+1}. In 2f, p^* is repaired by planning the valid blue segments to $\mathbf{T}_{\text{init}}, \mathbf{T}_{\text{desired}}$ and adding them to the start and end of p^* resulting in p_{N+1}.

Algorithm 1 Path retrieval

```
Input: Database $\mathcal{P}$, new assembly $r_{N+1}$, starting pose $\mathbf{T}_{\text{init}}$,
    goal pose $\mathbf{T}_{\text{desired}}$, task $\Theta$
Output: A path to reuse $p^*$ or $\varnothing$
 1: $[\mathbf{T}_{pi}^{\text{start}}, \mathbf{T}_{pi}^{\text{goal}}] \leftarrow$ eval_fk($\mathcal{P}, r_{N+1}$)
 2: $\delta_{1,i} \leftarrow$ calc_err($\mathbf{T}_{pi}^{\text{start}}, \mathbf{T}_{pi}^{\text{goal}}, \mathbf{T}_{\text{init}}, \mathbf{T}_{\text{desired}}$)
 3: $candidates \leftarrow$ select($\delta_{1,i}, K$)
 4: $\delta_{\text{min}}, p^* \leftarrow \infty, \varnothing$
 5: for $p$ in $candidates$ do
 6:     if $\neg$validate($p, \Theta, r_{N+1}$) then continue end if
 7:     $[\mathbf{q}_{N+1}^{ts}, \mathbf{q}_{N+1}^{tg}] \leftarrow$ solve_ik($r_{N+1}, \mathbf{T}_{\text{init}}, \mathbf{T}_{\text{desired}}$)
 8:     $[\mathbf{q}_{pi}^{s,\text{min}}, \mathbf{q}_{pi}^{g,\text{min}}] \leftarrow$
            find_nearest_config($p, \mathbf{q}_{N+1}^{ts}, \mathbf{q}_{N+1}^{tg}$)
 9:     $p \leftarrow$ crop($p, \mathbf{q}_{pi}^{s,\text{min}}, \mathbf{q}_{pi}^{g,\text{min}}$)
10:     $\delta_2 \leftarrow$ eval_cost($\mathbf{q}_{pi}^{s,\text{min}}, \mathbf{q}_{pi}^{g,\text{min}}, \mathbf{q}_{N+1}^{ts}, \mathbf{q}_{N+1}^{tg}$)
11:     if $\delta_2 < \bar{\delta}_2$ then continue end if
12:     if $\delta_2 < \delta_{\text{min}}$ then $[\delta_{\text{min}}, p^*] \leftarrow [\delta_2, p]$ end if
13: end for
```

components in black – at the start of p_i in orientation and translation, with $\Delta_{\Theta_{\text{R}}} \approx 90°$, and at the end, a pure translation of $\Delta_{r_{\text{sph}}}$.

2) Configuration Distance δ_2: After obtaining the K best candidate paths according to δ_1, we minimize the distance in configuration space to select the best path for reuse p^*. First, each of the K candidate paths p_i is validated (Alg. 1, line 6) for r_{N+1} to ensure that p_i does not collide with any obstacles in the configuration space of r_{N+1}. Colliding segments of p_i at either end are removed, as shown by the dotted line in Fig. 2b. Any p_i with a collision in its center is discarded.

At this point, the configurations at the ends of p_i are not moving the end-effector of r_{N+1} to the desired poses $\mathbf{T}_{\text{init}}$ and $\mathbf{T}_{\text{desired}}$ (red circles in Fig. 1). To minimize the length of repair paths, we find nearby inverse kinematic solutions for r_{N+1}. We take $\mathbf{q}_{pi}^{s}$ and $\mathbf{q}_{pi}^{g}$ (the configurations at either end of p_i) as initial guesses for the local inverse kinematics solver from [22], which, if successful, returns $\mathbf{q}_{N+1}^{ts}$, $\mathbf{q}_{N+1}^{tg}$ (Alg. 1, line 7). The black arrows in Fig. 2c indicate the local search.

Afterwards, we determine the nearest configurations by Euclidean distance on p_i to $\mathbf{q}_{N+1}^{ts}$ and $\mathbf{q}_{N+1}^{tg}$, denoted by $\mathbf{q}_{pi}^{s,\text{min}}$ and $\mathbf{q}_{pi}^{g,\text{min}}$ in line 8 of Alg. 1 and Fig. 2d. The candidate path p_i is cropped between $\mathbf{q}_{pi}^{s,\text{min}}$ and $\mathbf{q}_{pi}^{g,\text{min}}$, resulting in the final candidate to reuse; see removed red dashed line-segment in Fig. 2d.

Lastly, we define the second distance (see Fig. 2e) as

$$\delta_2 = \left\|\mathbf{q}_{pi}^{s,\text{min}} - \mathbf{q}_{N+1}^{ts}\right\|_2 + \left\|\mathbf{q}_{pi}^{g,\text{min}} - \mathbf{q}_{N+1}^{tg}\right\|_2. \qquad (2)$$

We reject p_i with δ_2 over a threshold $\bar{\delta}_2$ to avoid returning paths requiring substantial repairs. The path p^* and assembly r^* minimizing δ_2 are selected for path repair:

$$(p^*, r^*) = \underset{p,r \in \mathcal{P}}{\arg\min}(\delta_2(p, r)), \;\; s.t. \;\delta_2 < \bar{\delta}_2$$

We choose the first path if multiple paths with the same minimal distance exist.

To extend path retrieval to a set of $N_g > 1$ goals $\mathcal{G} = \{g^0, ..., g^{N_g}\}$, the start configuration $\mathbf{q}_{N+1}^{k-1}$ is fixed $\forall k \in]1, N_g]$ when planning the path from goal g^{k-1} to g^k, while $\mathbf{q}_{N+1}^{k}$ can still be any inverse kinematic solution of g^k for r_{N+1}. Paths are still retrieved when planning from one goal to the next, as described previously.

C. *Path Repair*

In this section, we describe path repair, as illustrated by the solid blue segments in Fig. 1 and 2f. Continuing from the result of path retrieval p^*, one needs to find two path segments connecting $\mathbf{q}_{pi}^{s,\min}$ and $\mathbf{q}_{pi}^{g,\min}$ directly to $\mathbf{q}_{N+1}^{ts}$ and $\mathbf{q}_{N+1}^{tg}$, respectively. RRT-Connect is used to plan these connecting path segments.

If both repairing paths are found, they are concatenated with the retrieved path p^*. This is the desired path p_{N+1} connecting the goals $\mathbf{T}_{\text{init}}$ and $\mathbf{T}_{\text{desired}}$ with the assembly r_{N+1}. As with path retrieval, in multi-goal situations, path repair needs to be applied to each p^* retrieved between two consecutive goals g^{k-1} and g^k.

D. *Integration into Modular Robot Optimization*

An application of our problem statement from Sec. I-B is modular robot optimization, e.g., using genetic algorithms [5]. The optimization tries to find the assembly and associated path, which minimizes a cost function C, such as cycle time, the negation of which $(-C)$ is the fitness function of the genetic algorithm. We integrate our novel algorithm from Sec. II-B and II-C as the planner to calculate this fitness. In this case, the database $\mathcal{P}$ stores valid paths of assemblies in a single genetic algorithm run, which our method adapts to assemblies in later generations.

When optimizing many assemblies, previous paths might be reused multiple times, resulting in multiple repairs to a path and, thereby, ever longer paths. Thus, we limit the number d_{p_i} of previous paths included in a path p_i. Only path-assembly pairs with d_{p_i} below a threshold value $\bar{d} \in \mathbb{N}_{>0}$ are added to the database $\mathcal{P}$.

In some cases, planning with reuse can fail for an assembly r_{N+1}. Therefore, we add *planning from scratch*, i.e., planning the complete path with RRT-Connect, as a fallback. Any path found by planning from scratch is also added to $\mathcal{P}$ together with r_{N+1}. Commonly, the fallback happens if $\mathcal{P}$ is empty at the start of optimization or the path retrieval/repair methods fail.

III. Numerical Experiments

A. *Setup*

We use the RRT-Connect implementation from the Open Motion Planning Library (OMPL) [23], and the modular robot simulator Timor Python [22]. Our baseline is planning from scratch, as presented in [5]. We compare the efficiency of this baseline and our algorithm by the planning time t_{plan} required to find a valid path.

The path planners are tested on tasks from the Composable Benchmark for Robotics Applications (CoBRA) [2], which provides tasks with different goals and obstacles. We select the tasks `Whitman2020/with_torque/{3g_3o/22, 3g_3o/25, 5g_5o/16, 5g_5o/45}`[5] for our experiments.

[5]Find each task by its ID on cobra.cps.cit.tum.de/tasks. The tasks contain three or five **g**oals and **o**bstacles. We omit the common prefix for brevity.

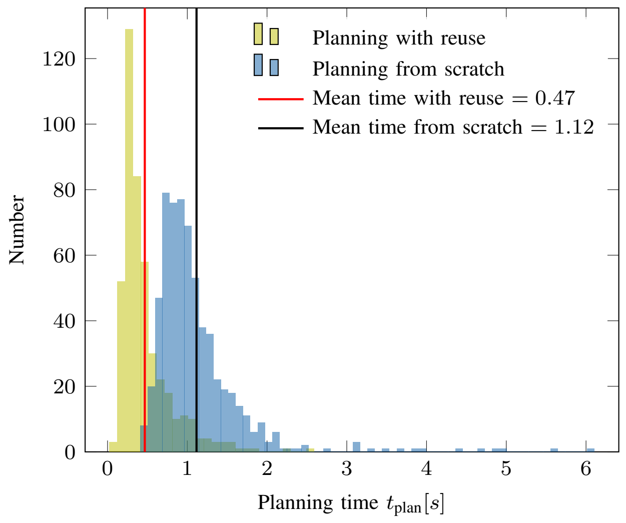


Fig. 3. Histogram of planning times t_{plan} for all assemblies with one goal in task `3g_3o/22`. The height of each bin indicates the number of assemblies with planning times within the corresponding range on the horizontal axis. Vertical lines mark the mean planning time for each planner.

Our first experiment in Sec. III-B plans paths between the first two goals of these four tasks. For each task, we find $\mathcal{A}_{\text{f}}$, a set of assemblies with valid inverse kinematics solutions at the goals of the task by applying the filters from [5]. We tested the path planners with a random selection of 40 assemblies from $\mathcal{A}_{\text{f}}$ and repeated each attempt 20 times, using planning with reuse and from scratch, respectively. The database $\mathcal{P}$ for reuse is initialized for each experiment with the paths found by planning from scratch in the same round. In this experiment, we limit the runtime of RRT-Connect in both methods to $5\,\text{s}$.

Next, the experiment in Sec. III-C considers path planning with multiple consecutive goals. Here, we sort the assembly set $\mathcal{A}_{\text{f}}$ used in Sec. III-B by regarding each assembly as a string of ordered modules and consider the first 40 assemblies; thereby, minimizing the difference between consecutive assemblies. Lastly, we test both planning methods within modular robot optimization based on a genetic algorithm in Sec. III-D.

B. *Planning with a Fixed Database and a Single Goal*

We first compare the performance of planning with reuse and from scratch for planning a single path between two goals, as shown in Fig. 1. Fig. 3 shows the histogram of planning times t_{plan} for task `3g_3o/22`; the distributions for the other tasks are quite similar. The average planning time for the tasks is shown in the first two rows of Tab. I. Overall, the planning time with the reuse method has a more concentrated distribution, as seen in Fig. 3. On average, over all tasks and runs, planning with reuse reduced the planning time by $50.58\,\%$.

Fig. 4 plots the success rate of both planners vs. timeout. In general, planning with reuse is more likely than planning from scratch to find a valid path at any timeout. We identify timeouts for the following experiments by a

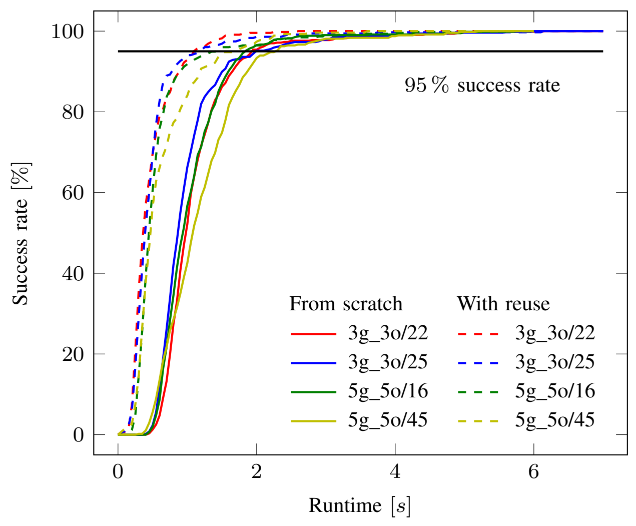


Fig. 4. The success rate of each planner and task versus its run time. The dashed lines represent the success rate of planning with reuse, and the solid lines for planning from scratch. Our desired success rate of 95 % is shown in black.

desired success rate of 95 %. The timeout for planning with reuse is, therefore, set to be 1.1 s (task `3g_3o/22`), 1.2 s (task `3g_3o/25`), 1.5 s (task `5g_5o/16`), and 1.7 s (task `5g_5o/45`); for planning from scratch to 2.0 s, 2.0 s, 1.9 s, and 2.3 s, respectively.

We assumed that the time to repair paths is proportional to the pose and configuration distance between the path ends and the goals of the task to motivate our distance functions. To validate this, we analyze the Pearson correlation coefficient[6] between all observed pose distances δ_1 or configuration distances δ_2 and planning times t_{plan}. We determine correlation coefficients with 95 % confidence intervals via bootstrapping within $[0.384, 0.500]$ and $[0.545, 0.652]$ for pose and configuration distance, respectively. Permutation testing rejects the hypothesis that there is no correlation in both cases with a p-value of 10^{-5}, i.e., the minimal possible given 10^5 resamples.

C. Planning with a Growing Database and Multiple Goals

This sub-section compares both planners in the multi-goal setting with an iteratively growing database, where planning with reuse can only retrieve its previous results. The time limits for both planners are set based on the time to reach a 95 % success rate in Fig. 4. We run the experiment on a server in parallel; thus, the absolute runtime increases in comparison to Sec. III-B.

First, we compare the planning time t_{plan}, as shown in the second half of Tab. I. Planning with reuse reduces it on average by 23.22 %. In 36.94 % of attempts the fallback from Sec. II-D is used. The average time to plan a path to a single goal only with reuse is 0.913 s. With the time limits determined in Sec. III-B, planning with reuse succeeds in 98.81 % of planning attempts, while planning from scratch succeeds in 97.28 %.

[6]Calculated using docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html, accessed on July 4th, 2024.

D. Integration into Modular Robot Optimization

We test planning with reuse inside the genetic algorithm introduced in Sec. II-D and compare it to planning from scratch, the previously used planner. For each task, we run the genetic algorithm six times with planning with reuse/from scratch embedded in the fitness function of the genetic algorithm to calculate the fitness $-C$. C is the execution time for a trajectory with limited joint velocities and accelerations such that a real robot can track them. Each found path is mapped to such a trajectory using [24].

We use the same initial population for the genetic algorithm in each experiment with either planning method. The initial population contains 120 individuals, evolving for 100 generations, resulting in 12 000 evaluated assemblies. Because our novel planning algorithm is designed to only test for feasible paths, we re-plan paths for the ten best assemblies found from scratch and report their minimum execution time as the final best cost C.

Tab. II summarizes the results of genetic algorithm optimization with both planners. Comparing the average planning time for each assembly t_{plan}, it is reduced by 6.96 % to 50.26 % when using planning with reuse instead of planning from scratch. The distribution of the planning time is stochastic, and the data lies in a big range, which is indicated by the given standard deviations. Post-processing is useful for our novel planning method, reducing the final cost versus the results during fitness evaluation by 19.41 %, making them on average only 5 % longer than the planning from scratch results. Lastly, we count the assemblies that successfully find paths and solve the tasks in the last two rows of Tab. II.

IV. Discussion

A. Standalone Comparison of Planning Methods

The results in Sec. III-B and Sec. III-C both indicate that planning with reuse outperforms planning from scratch in planning time t_{plan}. Compared to the multi-goal results, the single-goal results in Sec. III-B show a larger planning time reduction and shorter average planning time. This could be due to a better database $\mathcal{P}$ where all entries are planned from scratch in the single-goal setting. Additionally, the fallback to planning from scratch increases the average runtime of planning with reuse in the multi-goal setting. Lastly, in the multi-goal setting, only the configurations of the next goal pose can be chosen freely, resulting in a longer configuration distance to repair from $\mathbf{q}_{pi}^{s,\min}$ to $\mathbf{q}_{N+1}^{k-1}$.

B. Comparison within Modular Robot Optimization

Sec. III-D shows that planning with reuse reduces the average planning time t_{plan} for a single assembly. As we run these optimizations on a server in parallel, the absolute runtime increases compared to Sec. III-B. The relative time reduction is larger than in Sec. III-C. We suggest that compared to the assemblies considered in Sec. III-C, the genetic algorithm generates assemblies with increased similarity,

TABLE I

PLANNING TIMES FOR PLANNING WITH REUSE (PWR) AND PLANNING FROM SCRATCH (PFS) FOR SINGLE AND MULT. GOALS (AVG. ± STD. DEV.). **BOLD VALUES** INDICATE THE MINIMUM TIME PER TASK, CONSIDERED GOAL(S), AND MEASUREMENT.

Goals	Subpart	Algorithm	`3g_3o/22`	`3g_3o/25`	`5g_5o/16`	`5g_5o/45`
Single	-	PWR	$(\mathbf{0.47 \pm 0.32})\,\mathbf{s}$	$(\mathbf{0.50 \pm 0.48})\,\mathbf{s}$	$(\mathbf{0.58 \pm 0.53})\,\mathbf{s}$	$(\mathbf{0.63 \pm 0.48})\,\mathbf{s}$
	-	PFS	$(1.12 \pm 0.60)\,\mathrm{s}$	$(1.02 \pm 0.61)\,\mathrm{s}$	$(1.06 \pm 0.55)\,\mathrm{s}$	$(1.20 \pm 0.66)\,\mathrm{s}$
Multiple	Reuse	PWR	$(0.93 \pm 0.71)\,\mathrm{s}$	$(0.76 \pm 0.58)\,\mathrm{s}$	$(0.91 \pm 0.68)\,\mathrm{s}$	$(1.05 \pm 0.74)\,\mathrm{s}$
	Fallback	PWR	$(1.77 \pm 0.70)\,\mathrm{s}$	$(1.60 \pm 0.70)\,\mathrm{s}$	$(1.82 \pm 0.74)\,\mathrm{s}$	$(1.82 \pm 0.84)\,\mathrm{s}$
	Total	PWR	$(\mathbf{1.34 \pm 0.82})\,\mathbf{s}$	$(\mathbf{1.06 \pm 0.74})\,\mathbf{s}$	$(\mathbf{1.18 \pm 0.81})\,\mathbf{s}$	$(\mathbf{1.30 \pm 0.85})\,\mathbf{s}$
	-	PFS	$(1.53 \pm 0.57)\,\mathrm{s}$	$(1.43 \pm 0.56)\,\mathrm{s}$	$(1.69 \pm 0.62)\,\mathrm{s}$	$(1.72 \pm 0.76)\,\mathrm{s}$

TABLE II

RESULTS FOR PLANNING WITH REUSE (PWR) AND PLANNING FROM SCRATCH (PFS) USED INSIDE THE GENETIC ALGORITHM (AVG. ± STD. DEV.).

Observation	Algorithm	`3g_3o/22`	`3g_3o/25`	`5g_5o/16`	`5g_5o/45`
Planning time t_{plan}	PWR	$(\mathbf{3.09 \pm 2.13})\,\mathbf{s}$	$(\mathbf{2.87 \pm 2.25})\,\mathbf{s}$	$(\mathbf{7.56 \pm 4.68})\,\mathbf{s}$	$(\mathbf{10.02 \pm 3.13})\,\mathbf{s}$
	PFS	$(4.97 \pm 1.24)\,\mathrm{s}$	$(5.77 \pm 1.26)\,\mathrm{s}$	$(10.01 \pm 3.08)\,\mathrm{s}$	$(10.77 \pm 2.01)\,\mathrm{s}$
Final path cost C	PWR	$(13.51 \pm 2.70)\,\mathrm{s}$	$(13.64 \pm 3.44)\,\mathrm{s}$	$(25.49 \pm 4.71)\,\mathrm{s}$	$(28.80 \pm 7.26)\,\mathrm{s}$
	PFS	$(\mathbf{12.89 \pm 3.57})\,\mathbf{s}$	$(\mathbf{13.24 \pm 4.45})\,\mathbf{s}$	$(\mathbf{24.31 \pm 3.31})\,\mathbf{s}$	$(\mathbf{26.15 \pm 4.07})\,\mathbf{s}$
Planning successes	PWR	(106.67 ± 132.27)	(120.00 ± 164.47)	(19.83 ± 20.31)	(7.00 ± 1.55)
	PFS	(102.17 ± 125.78)	(90.83 ± 107.62)	(20.33 ± 23.91)	(8.17 ± 3.13)

as it mostly recombines previous ones with crossover and mutation. Additional time savings are realized for identical assemblies appearing in different generations.

In easier tasks, i.e., those with three goals and three obstacles, the genetic algorithm will find more feasible assemblies, resulting in a larger database of previous paths $\mathcal{P}$ and, thereby, increasing the advantage of planning with reuse, as shown in the left columns of Tab. II. The average planning time for the `3g_3o` tasks is reduced by about $44\,\%$, while the `5g_5o` tasks only saw a $16\,\%$ decrease.

According to Tab. II, only a few assemblies can solve each task. Even in the best case, paths were found for less than $1\,\%$ of assemblies. The other assemblies usually cannot pass the filter requiring a valid inverse kinematics solution for every goal, contributing to the overall runtime of the genetic algorithm. Still, path planning accounts for a relatively high fraction of overall genetic algorithm runtime ($17.46\,\% - 30.19\,\%$ with planning from scratch; $15.54\,\% - 25.84\,\%$ with planning with reuse). Also, increased overall population fitness requires more path planning, increasing the share of planning time t_{plan} of total runtime. In most cases, planning with reuse shortens the average total time of genetic algorithms. This reduced optimization time does not affect the final optimization output, as indicated by the comparable best path costs found with both planning methods.

V. CONCLUSION

We propose planning with reuse – a novel method for modular robot path planning. The method is inspired by the slight changes to modular robots during common iterative optimization algorithms, such as genetic algorithms. Our method finds paths for new module assemblies by retrieving a path planned for a previously considered assembly and adapting it to the new assembly.

Compared to planning from scratch, planning with reuse exhibits a reduced average planning time for all tasks, whether planning for a single or multiple goals. Integrating our approach in optimizing assemblies of modular robots using genetic algorithms results in decreased planning times, while the final solutions from both approaches have comparable cycle times.

ACKNOWLEDGEMENTS

Firstly, we thank Jonathan Külz for his great collaboration on the Timor Python toolbox. Secondly, we would like to thank Ang Li, Dian Yuan, Melike Keskin, and Yulia Nikirova, students at the Technical University of Munich, who contributed to the reuse-repair planner and genetic algorithm code bases. Lastly, we acknowledge financial support from the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant number AL 1185/31-1.

REFERENCES

[1] 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.

[2] M. Mayer, J. Külz, and M. Althoff, “CoBRA: A composable benchmark for robotics applications,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 17 665 – 17 671.

[3] E. Icer, A. Giusti, and M. Althoff, “A task-driven algorithm for configuration synthesis of modular robots,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5203–5209.

[4] S. B. Liu and M. Althoff, “Optimizing performance in automation through modular robots,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4044–4050.

[5] J. Külz and M. Althoff, “Optimizing modular robot composition: A lexicographic genetic algorithm approach,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 16 752–16 758.

[6] J. Whitman and H. Choset, “Task-specific manipulator design and trajectory synthesis,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 301–308, 2019.

[7] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, “Modular robot design synthesis with deep reinforcement learning,” in *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020, pp. 10 418–10 425.

[8] J. Hu, J. Whitman, M. J. Travers, and H. Choset, “Modular robot design optimization with generative adversarial networks,” in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2022, pp. 4282–4288.

[9] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane, “Computational design of robotic devices from high-level motion specifications,” *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1240–1251, 2018.

[10] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[11] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[12] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Computer Science Dept., Iowa State University, Tech. Rep., 1998.

[13] J. J. Kuffner and S. M. La Valle, “RRT-Connect: An efficient approach to single-query path planning,” in *Proceedings - IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 995–1001.

[14] M. Vande Weghe, D. Ferguson, and S. S. Srinivasa, “Randomized path planning for redundant manipulators without inverse kinematics,” *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, pp. 477–482, 2007.

[15] W. Wang and Y. Li, “Path planning for redundant manipulator without explicit inverse kinematics solution,” *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1918–1923, 2009.

[16] D. Berenson, P. Abbeel, and K. Goldberg, “A robot path planning framework that learns from experience,” in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3671–3678.

[17] A. Dobson and K. E. Bekris, “Sparse roadmap spanners for asymptotically near-optimal motion planning,” *The International Journal of Robotics Research*, vol. 33, pp. 18 – 47, 2014.

[18] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, “Memory of motion for warm-starting trajectory optimization,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.

[19] C. Chamzas, A. Shrivastava, and L. E. Kavraki, “Using local experiences for global motion planning,” in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8606–8612.

[20] C. Chamzas, A. Cullen, A. Shrivastava, and L. E. Kavraki, “Learning to retrieve relevant experiences for motion planning,” in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7233–7240.

[21] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[22] J. Külz, M. Mayer, and M. Althoff, “Timor Python: A toolbox for industrial modular robotics,” in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 424–431.

[23] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[24] T. Kunz and M. Stilman, “Time-optimal trajectory generation for path following with bounded acceleration and velocity,” in *Robotics: Science and Systems*, 2012, pp. 9–13.