Timor Python: A Toolbox for Industrial Modular Robotics

Jonathan Külz, Matthias Mayer, and Matthias Althoff

***Abstract*— Modular Reconfigurable Robots (MRRs) represent an exciting path forward for industrial robotics, opening up new possibilities for robot design. Compared to monolithic manipulators, they promise greater flexibility, improved maintainability, and cost-efficiency. However, there is no tool or standardized way to model and simulate assemblies of modules in the same way it has been done for robotic manipulators for decades. We introduce the Toolbox for Industrial Modular Robotics (Timor), a Python toolbox to bridge this gap and integrate modular robotics into existing simulation and optimization pipelines. Our open-source library offers model generation and task-based configuration optimization for MRRs. It can easily be integrated with existing simulation tools – not least by offering URDF export of arbitrary modular robot assemblies. Moreover, our experimental study demonstrates the effectiveness of Timor as a tool for designing modular robots optimized for specific use cases.**

I. Introduction

Modular Reconfigurable Robots (MRRs) offer a promising extension to conventional robotic manipulators. Assembled from a combination of individual hardware parts referred to as modules, they pose unprecedented flexibility, easy maintainability, and transportability by design [1], [2]. However, the vast number of possible robot morphologies resulting from a given set of modules comes with significant challenges when designing MRRs [3]. Every configuration of modules leads to a new dynamic and kinematic model, which must be generated to apply control and planning procedures designed for traditional manipulators. While there are multiple well-established frameworks for conventional robots enabling physical simulation, none of them can be used in a scenario where the robot model is subject to constant change.

A. Contributions

The **T**oolbox for **I**ndustrial **Mo**dular **R**obotics (Timor) provides modeling and simulation capabilities for MRRs, starting from a standardized module description. Assembled configurations of modules can easily be exported to the Unified Robot Description Format (URDF)[1] for easy integration into existing pipelines. Furthermore, it facilitates the design of task-tailored MRRs by incorporating search heuristics and optimization algorithms. User-defined as well as provided module libraries can easily be assembled to robots with the possibility to automatically generate dynamic, kinematic, and collision models, all within the same framework. Furthermore, any module combination – also referred to as *assembly* – comes with visualization capabilities. Timor is fully implemented in Python and hosted on `https://gitlab.lrz.de/tum-cps/timor-python`. Code coverage, unit tests, documentation, and a moderated issue board are publicly available. The toolbox and all dependencies can be installed with

```
$ pip install timor-python
```

B. Related Work

Timor enables modeling, simulating, and designing arbitrary configurations of robot modules, effectively bridging the divide between modular robots and traditional manipulators. As a result, recent advancements in robotic simulation suites are now within reach for MRRs, taking an essential step toward their implementation in industrial settings.

a) Modular Reconfigurable Robots (MRRs): An MRR consists of multiple independent modules that can be reconfigured externally or by the robot (self-reconfiguring modular robots). The separation of module-level and system-level design enables rapid change-over, expansion, and robustness [1], [4]. Early publications on the concept of MRRs, such as the CEBOT [5] and the RMMS [6] in 1988, already established a distinction between joint and link modules with the possible extension of functional (i.e., end effector) modules, a classification that is frequently used until today. Over the last decades, various modular robot systems have been proposed [2], [7], [8]. However, it is only recently that MRRs have finally left research laboratories.

The benefits of MRRs become particularly noteworthy when multiple tasks, which may not be known during the hardware design phase, need to be accomplished using a single robot or set of modules only [1]. However, utilizing modular robots in varying configurations often requires the expertise of professional and experienced programmers, rendering industrial implementation impractical. This limitation has been overcome with the introduction of self-programming capabilities for modular robots [9]. In recent years, multiple industrial solutions for modular robots entered the market[2,3] or have been announced for a future release[4]. Despite the increased supply in MRR hardware and the acknowledgment of the necessity of fast model generation [10], distinct modeling libraries for industrial MRRs are not yet available and development and research

All authors are with the Cyber-Physical Systems Group, Department of Computer Science, Technical University of Munich, 85748 Garching, Germany `[jonathan.kuelz, matthias.mayer, althoff]@tum.de`. Matthias Althoff and Jonathan Külz are also with the Munich Center for Machine Learning (MCML).

[1]`https://wiki.ros.org/urdf`

[2]`https://www.hebirobotics.com/`

[3]`https://www.robco.de/en`

[4]`https://www.beckhoff.com/en-en/products/motion/atro-automation-technology-for-robotics/`

departments often resort to tools designed for traditional robotics.

Theoretical groundwork for MRR model generation have been published in recent years: The CoBRA benchmark [11] provides a standardized module description format and benchmarks to compare MRRs and the authors of [12], [13] propose a language to describe MRRs through constraints implicitly. Despite introducing a framework for an MRR modeling language, these works lack the possibility of kinematic and dynamic model generation for explicitly defined configurations of robot modules. Modular robot URDF files can be generated using a SolidWorks plugin as described in the work in [14]. However, its limitations include dependence on third-party software and a need for pre-defined CAD models. The work in [15] proposes a procedure for automatic robot model description generation but only supports serially connected and asymmetric modules.

Timor extends existing theoretical works by offering support of kinematic trees and a unified module description format to model modules with an arbitrary number of connection interfaces, bodies, and joints. It further implements this approach and leverages it for MRR design, making it accessible for future research.

b) Search for MRR configurations: Identifying an optimized configuration of MRR modules to meet desired robot performance characteristics poses a considerable challenge. An exhaustive algorithm that uses enumeration of kinematically distinct configurations is theoretically applicable [16]; however, the large search space renders this method impractical. Various attempts have been made to simplify this problem: In [9], [17], the search is limited to configurations with constraints on the alternation between static links and joints. The work in [18] restricts the search even further to kinematics known from industrial robots. In addition to a search space reduction, heuristic algorithms are commonly applied. The early work in [19] co-adapted morphology and control of virtual creatures using genetic algorithms (GAs). In GAs, solution candidates (*chromosomes*) composed of a fixed number of variables (*genes*) are evolved for a fixed number of generations by selection, reproduction, and mutation operations [20]. In applications of MRR design, a representation of modules as genes and assemblies as chromosomes is usually utilized. This approach has ever since been explored in various works [21], [22], [23]. Furthermore, learning-based methods have recently been proposed to optimize modular agents both in industrial and non-industrial contexts [24], [25], [26]. While these approaches share a common robot design principle and are evaluated in simulation, they are based on individual software interfaces. There is no possibility to evaluate them on a common set of robot modules, nor are they directly applicable to custom industrial tasks. Timor provides a unified interface for constructing, evaluating, and optimizing MRR models, allowing one to utilize arbitrary optimization algorithms in a plug-and-play manner.

c) Robotics Simulation: Simulation tools offer great potential for developing control, morphology, and motion planning in robotics [27]. There is a wide range of sophisticated physics engines like Bullet [28], MuJoCo [29], DART [30], or the Gazebo Simulator [31] that can be used to develop and fine-tune model-based controllers and precisely simulate robot and contact dynamics. On the other side of the spectrum are toolboxes with a more narrow focus on kinematic and dynamic modeling in robotics, such as the Robotics Toolbox [32] or Pinocchio [33]. These tools, frequently used for tasks such as path planning and controller synthesis, commonly use a higher level of abstraction that realizes fast computation and provide a high-level application programming interface (API). While there is a great choice of tools for monolithic robots, all assume a known robot model. A lack of support for MRRs, whose robot model is subject to change, requires users to craft workarounds for established simulation tools, mostly tailored to specific hardware. Simulators like the USSR [34], ReBots [35], or VisibleSim [36] enable the simulation of self-reconfigurable robots that consist of up to millions of modules. However, these simulators do not provide a high-fidelity computation of robot kinematics and dynamics, which is essential for achieving realistic simulations of industrial tasks. There have been efforts to connect disassembled robot modules with traditional robotics libraries, but they are only applicable to specific sets of modules, such as the HEBI HRDF format[5]. Timor aims to provide a standardized interface to enable the modeling of arbitrary module libraries, making it possible to use established tools for MRRs.

II. Model Generation

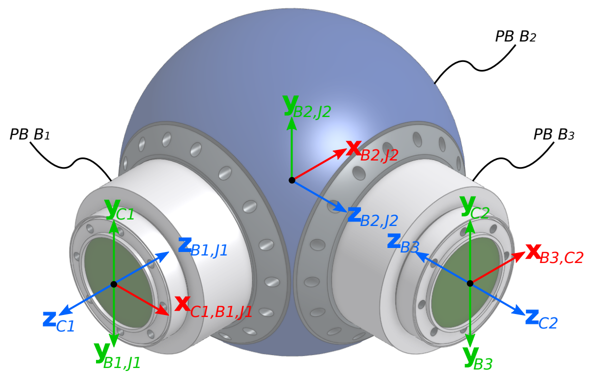
Timor provides automatic model generation for module configurations, including a URDF export interface. In addition, the models are used by the built-in simulation capabilities of Timor to support the design process of MRRs.

A. Graph Representations

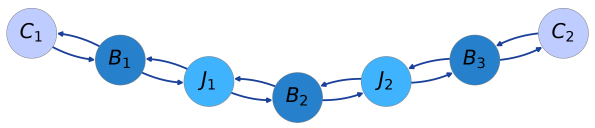
Timor can compute two graph representations for robot modules and the relations between them. They are generated automatically without additional user input and serve as an abstract and domain-independent view of MRRs. A short overview of the terminology is given in the following – details on the definition and generation of the graphs are provided in the referenced sections.

- The directed *module graph* G_m (Sec. II-B) represents a single module and, on its edges, contains information about the relative transformations between the reference frames of its constituent joints, bodies, and connectors.
- The directed *assembly graph* G_a (Sec. II-E) is a combination of multiple module graphs. It represents the relative placement of every joint, body, and connector reference frame within an assembly of modules.

As a consequence of declaring the graphs directed, homogeneous transformations between their constituent elements can directly be assigned as edge features.



(a) Powerball module with reference frames F for connectors (F_{C1} and F_{C2}), bodies (F_{B1}, F_{B2}, and F_{B3}), and revolute joints (F_{J1} and F_{J2}).



(b) Module graph G_m for the Powerball module. The nodes of the graph represent bodies (B_i), joints (J_i), or connectors (C_i). The homogeneous transformations between their reference frames are stored on the edges of the graph.

```
<robot name="IMPROV">
  <link name="PB B1">
    <!-- Link Details -->
  </link>
  <link name="PB B2">
    <!-- Link Details -->
  </link>
  <link name="PB B3">
    <!-- Link Details -->
  </link>

  <!-- Data from other modules -->

  <joint name="PB_conector_in" type="fixed">
    <parent link="base body"/>
    <!-- Joint Details -->
  </joint>
  <joint name="PB_joint_1" type="revolute">
    <parent link="PB B1"/>
    <child link="PB B2"/>
    <!-- Joint Details -->
  </joint>
  <joint name="PB_joint_2" type="revolute">
    <parent link="PB B2"/>
    <child link="PB B3"/>
    <!-- Joint Details -->
  </joint>
</robot>
```

(c) The URDF excerpt shows the structure for links and joints modeling the Powerball module in an assembly, assuming it is attached to a base body.

Fig. 1. Relative placements of bodies, connectors, and joints in a module are defined by the positioning of their reference frames. The homogeneous transformations between reference frames are stored in the module graph G_m that also captures neighbor relations. This information can be used to auto-generate URDF models for assemblies.

B. Module Definition

The implementation of modules in Timor is inspired by the framework introduced for the CoBRA benchmark [11]. Modules are composed of rigid bodies and joints. Bodies can have an arbitrary number of connectors, defining interfaces to attach them to other modules. Every body, joint, and connector is assigned a reference frame F. The placement and orientation of any joint or connector reference frame F_i is defined relative to a body reference frame F_j by a homogeneous transformation $T(F_i, F_j)$ – in this case, we say there exists a neighbor relationship between the corresponding elements. Fig. 1a shows a module designed for the Schunk LWA 4P robot, consisting of three bodies, two connectors, two revolute joints, and the corresponding reference frames.

For any module m_i with connectors $\mathcal{C}_{m_i}$, bodies $\mathcal{B}_{m_i}$, and joints $\mathcal{J}_{m_i}$, we define the directed *module graph* as a tuple $G_{m_i} = (\mathcal{V}_{m_i}, \mathcal{E}_{m_i})$ with

$$\mathcal{V}_{m_i} = \mathcal{C}_{m_i} \cup \mathcal{B}_{m_i} \cup \mathcal{J}_{m_i}$$
$$\mathcal{E}_{m_i} = \{(u, v) \in \mathcal{V}_{m_i} \times \mathcal{V}_{m_i} \mid \texttt{neighbor}(u, v)\},$$

where the $\texttt{neighbor}(u, v)$ predicate evaluates to true if a neighbor relationship between u and v exists and false, otherwise. To any edge $e \in \mathcal{E}_{m_i}$ between vertices (u, v) with reference frames (F_u, F_v), we assign the feature $T_e := T(F_u, F_v)$. Fig. 1b shows the module graph for the Powerball module displayed in Fig. 1a.

While a module can be composed of an arbitrary number of these base elements, it usually represents one piece of hardware as produced by a manufacturer. Due to the possible multiplicity of bodies and joints, even complex hardware with arbitrary geometries, such as fully integrated multiple degrees-of-freedom joints or mobile bases can easily be described as a single module.

C. Connectors

Connectors are interfaces placed on bodies, used to define a connection between two modules. Any connector C_i has a *gender* $g_i \in \{$**m**ale, **f**emale, **h**ermaphroditic$\}$, a *type* t_i, and a *size* s_i. There are two special connector types, *eef* and *base*, that define end effector frame(s) and the base reference frame(s). Apart from these reserved keywords, other types can be chosen freely by users to reflect the connector hardware (e.g., *flange*, *clamp*). We define the compatibility function between two connectors (C_i, C_j) as

$$\sigma_c(C_i, C_j) := \begin{cases} \text{true}, & \begin{aligned} &\text{if } t_i = t_j, \\ &\text{and } s_i = s_j, \\ &\text{and} \begin{cases} g_i \neq g_j, \text{ if } g_i, g_j \in \{m, f\}, \\ g_i = g_j = h, \text{ otherwise.} \end{cases} \end{aligned} \\ \text{false, otherwise.} \end{cases}$$

[5]https://github.com/HebiRobotics/hebi-hrdf

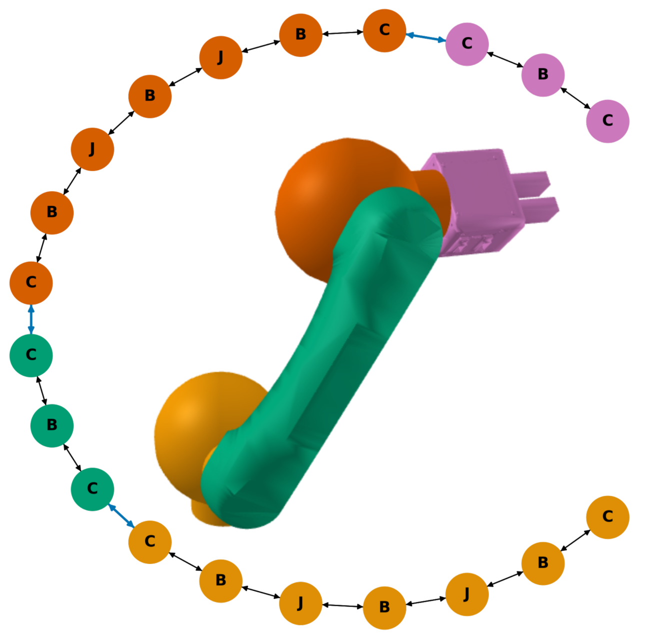


Fig. 2. A Schunk LWA 4p robot built from four modules and the corresponding assembly graph G_a. The nodes of the composing module graphs G_m are drawn in the same color as the corresponding modules. Edges connecting modules (Σ_d) are highlighted in blue.

Similarly, we define the compatibility function between two modules (m_1, m_2) with connectors $(\mathcal{C}_{m_1}, \mathcal{C}_{m_2})$ as

$$\sigma_m(m_1, m_2) := \begin{cases} \text{true, if } \exists (C_i, C_j) \in \mathcal{C}_{m_1} \times \mathcal{C}_{m_2} : \sigma_c(C_i, C_j) \\ \text{false, otherwise.} \end{cases}$$

The possibility to assign an arbitrary number of connectors to a body and the introduction of hermaphroditic connectors allows one to model non-serial kinematics and (multidirectional) modules without a unique mounting orientation. Without loss of generality, we assume that the reference frames (F_i, F_j) of the connectors (C_i, C_j) when connected are arranged such that the x-axes are aligned, and the z-axes are pointing away from the module (Fig. 1a). Therefore, $T(F_i, F_j) = R_x(\pi)$ where $R_x(\pi)$ is a homogeneous transformation performing a rotation of 180° around the x-axis.

D. Assembling Modules

We denote a set of robot modules as $\mathcal{R}$. A configuration of modules $M = \{m_1, \dots, m_n\}$ with connections $\Sigma = \{\{C_a, C_b\}, \{C_c, C_d\}, \dots\}$ is referred to as an assembly $\mathcal{A}$. Timor efficiently depicts both serial and branched kinematics. For representing closed-chain kinematics, Timor provides a robust framework, although simulation and validity checks require utilization of external libraries. Recognizing that many MRR configurations are chains, Timor also allows specifying an assembly implicitly as a tuple of modules $(m_1, m_2, \dots)$ as long as there is exactly one possible connection between any two neighboring modules (m_i, m_{i+1}).

Algorithm 1 Generate URDF

```
 1: function GENERATE_URDF($Assembly, name$)
 2:     $G \leftarrow Assembly$ graph $G_a$
 3:     $C_b \leftarrow Assembly$ base connector
 4:     $urdf \leftarrow$ initialize URDF($name$)
 5:     $T \leftarrow \{C_b.frame : \mathbb{I}_4\}$            ▷ $\mathbb{I}_4 \leftarrow$identity matrix
 6:     for (node $n$, edge $e$, successor $s$) $\leftarrow$ bfs($G, C_b$) do
 7:         $F_n, F_s \leftarrow n.frame, s.frame$
 8:         $T_s \leftarrow T[F_n] \cdot e.T$
 9:         $T[F_s] \leftarrow T_s$
10:         if is body($s$) then
11:             $link \leftarrow$ make URDF link $(s, T_s)$
12:             $urdf \leftarrow$ append($urdf, link$)
13:         else if is joint($s$) then
14:             $joint \leftarrow$ make URDF joint $(s, T_s)$
15:             $urdf \leftarrow$ append($urdf, joint$)
16:             $T[F_s] \leftarrow \{F_b : \mathbb{I}_4\}$
17:         else if is connector($s$) then
18:             $joint \leftarrow$ make URDF fixed joint $(s, T_s)$
19:             $urdf \leftarrow$ append($urdf, joint$)
20:             $T[F_s] \leftarrow \{F_b : \mathbb{I}_4\}$
21:         end if
22:     end for
23: end function
```

E. Building Kinematic and Dynamic Models

For deriving the kinematic and dynamic model of an assembly, we introduce the directed *assembly graph* (Fig. 2) as the tuple $G_a = (\mathcal{V}_a, \mathcal{E}_a)$ with

$$\mathcal{V}_a = \mathcal{V}_{m_1} \cup \dots \cup \mathcal{V}_{m_n},$$
$$\mathcal{E}_a = \Sigma_d \cup \mathcal{E}_{m_1} \cup \dots \cup \mathcal{E}_{m_n},$$

where Σ_d represents the connections between different modules and contains, for every connection in Σ, two directed, antiparallel edges (Fig. 2) to which we assign the feature $T_e = T_e^{-1} := R_x(\pi)$.

For any given sequence $E(u, v) = (e_1, \dots, e_n)$ of edges on a path between two nodes (u, v) in G_a, the homogeneous transformation between any two reference frames $T(F_u, F_v)$ is

$$T(F_u, F_v) = T_{e_1} T_{e_2} \dots T_{e_n}, \qquad u \neq v$$

Obviously, any sequence E works; we usually determine the shortest one using breadth-first search.

By performing a breadth-first iteration over G_a, starting at the base reference frame, Timor generates URDF descriptions for arbitrary module arrangements, as shown in Alg. 1, as long as G_a composes a tree (URDF does not natively support closed-chain kinematics). Homogeneous transformations from parent joints to URDF elements are stored in the map T, so $T[F]$ is the relative transformation between a frame and its parent joint. All further required information to write a corresponding URDF element[6,7] such

[6]URDF links: `https://wiki.ros.org/urdf/XML/link`
[7]URDF joints: `https://wiki.ros.org/urdf/XML/joint`

```
import numpy as np
from timor.Module import *
from timor.utilities.visualization import animation

db = ModulesDB.from_json_file(db_file)
modules = ('base', 'J2', 'i_45', 'J2', 'J2', 'eef')
A = ModuleAssembly.from_serial_modules(db, modules)
q0 = A.robot.random_configuration()
q1 = A.robot.random_configuration()
trajectory = np.linspace(q0, q1)
animation(A.robot, trajectory, dt=.1)
```

Fig. 3. Code sample to generate an animation.

as inertia or joint limits are stored within the modules. Fig. 1c shows the resulting structure for a partial URDF generated for the module shown in Fig. 1a.

The generation of a dynamic and kinematic model from URDF that can be used for simulation can be achieved by various software packages, such as the open-source tool Pinocchio [33], [37]. For the sake of computational efficiency, we also provide a direct conversion from any assembly to Pinocchio, omitting an intermediate URDF generation. Furthermore, Timor offers an interface to the FCL library [38] that we use for efficient collision checking.

III. MRR CONFIGURATION SEARCH

As the number of possible MRR configurations grows exponentially with the number of assembled modules, the search for an optimal configuration for an application cannot be performed exhaustively in general. Timor offers multiple tools to aid human and algorithmic optimization of assemblies for robot tasks.

A. Task Definition

Tasks can be formally described in the format introduced in [11]. They are composed of goals that can be end-effector poses to reach or trajectories to follow. All tolerances, obstacles, and constraints supported by the CoBRA benchmark can be specified and visualized with Timor, thus supporting a wide range of applications.

B. Visualization

Timor extends the meshcat[8] visualizer capabilities that are integrated in Pinocchio to enable manual inspection in an interactive browser-based visualization for robots and tasks. Furthermore, Timor allows generating videos for robot trajectories and exporting them as a file. Fig. 3 shows a code sample for loading a set of modules, defining an assembly, generating the corresponding robot model, and visualizing a random trajectory.

C. Pruning Iterators

Timor can enumerate valid combinations between modules and can produce configurations satisfying human-specified constraints without the need of pre-computing all valid configurations, therefore being memory-efficient. In particular, reasonable alternations between static links and joints, as proposed in [9], [17], can be enforced – not at least to resemble kinematics from industrial robots as in [18]. Furthermore, a custom set of valid sequences of modules can be defined explicitly, and the range of desired degrees of freedoms for an MRR can be set.

D. Assembly Filters

In the search for task-specific MRRs, the feasibility of subproblems is often checked. Simple metrics can be used to eliminate a majority of unfit MRR morphologies before performing computationally expensive evaluations [22]. Timor implements *filters* on criteria that are first evaluated for the goal pose and later for an entire trajectory, such as

- kinematic solutions exist,
- no self-collision, or
- no collision with the environment.

By evaluating metrics prior to model generation whenever feasible, the benefits of having a comprehensive and integrated library are demonstrated.

E. Genetic Algorithms (GAs)

The flexibility of MRRs can be leveraged to mimic various industrial robot kinematics. However, overly strong constraints on the search space for MRR configurations can limit the potential to discover specialized, previously unseen kinematics. Metaheuristics, such as GAs, offer a potential solution by providing a method to explore a range of MRR configurations while efficiently navigating the search space. Timor provides a user-friendly interface to integrate GAs into the optimization process of MRR configurations. Users can define industrial tasks, load a set of available modules, and optimize their configuration using GAs within minutes. As demonstrated in Sec. IV-B, these capabilities can be utilized to uncover unique robot morphologies.

IV. NUMERICAL EXPERIMENTS

To demonstrate the usefulness of Timor, we conducted two experiments. In our first experiment, we use a module set that was published in [9] to generate URDF models for arbitrary combinations of modules, showing how Timor can be leveraged to extend and unify existing work on MRRs. In our second experiment, we compare two optimization algorithms to human experts in designing a use-case-tailored MRR. The code to reproduce both experiments and more details on experimental results are provided together with a set of tutorials in the official repository[9]. All experiments were conducted on a desktop PC with an Intel i7-11700KF processor.

[8]https://github.com/rdeits/meshcat-python

[9]https://gitlab.lrz.de/tum-cps/timor-python/-/tree/main/tutorials

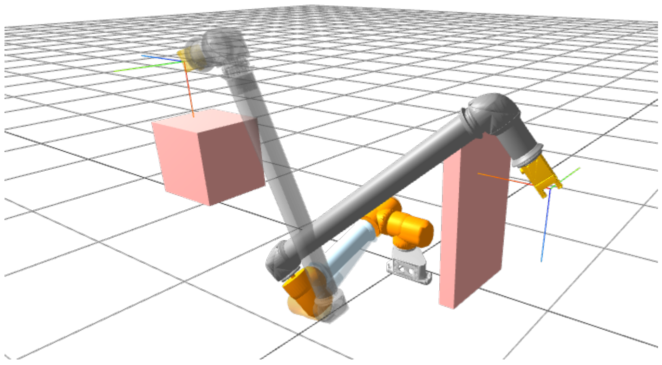


Fig. 4. Best lightweight assembly: The lightest robot reaching the two goals (marked as coordinate systems) while avoiding obstacles (red) was found by a GA.

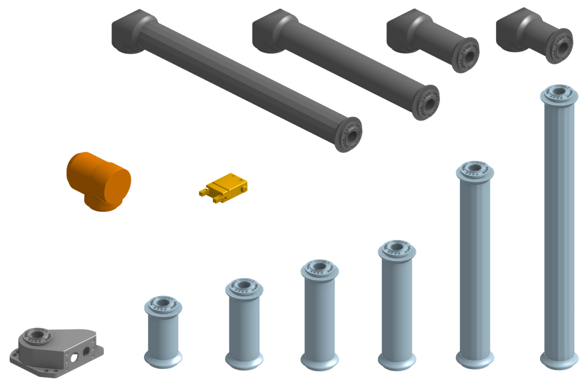


Fig. 5. Modules of the user study: For the experiment, we provide ten static link modules (grey) and one module containing a revolute joint (orange). The base and end effector are automatically added to every assembly.

A. Model Generation for an Existing Module Set

For this experiment, we use the IMPROV module set as introduced in [9], which is composed of modules of the Schunk LWA 4P robot and additionally designed modules. It contains two Powerballs (shown in Fig. 1a), ten static links of varying sizes and shapes, and a base module. By applying Timor's *assembly iterator*, we obtain the 32,768 possible configurations of modules for a six degrees-of-freedom robot as reported in [9]. Generating URDF files for all of them takes 127 seconds (3.9ms on average). Directly transforming an assembly to a Pinocchio-based robot model is equally computationally expensive. By providing tens of thousands of kinematic and dynamic models in a standardized format within a matter of minutes, Timor facilitates research on MRRs.

B. User Study

In a simulated environment, we challenged human experts to optimize the configuration of an MRR given the task in Fig. 4 and compared their results to two algorithms in Timor. A common use case for modular robots is machine tending, where the robot has to perform a pick-and-place operation in a potentially cluttered environment. We abstract this task by fixing a robot base in an environment with two static obstacles and two goal poses that need to be reached. An assembly poses a valid solution if an inverse kinematics solution can be found for both goals, given a position tolerance of 0.01mm and an orientation tolerance of 45° around an arbitrary axis relative to the desired positioning. While path planning was not explicitly incorporated into the optimization process, we could identify valid trajectories for animating both the algorithm and human-generated results by adopting basic sampling-based planners.

For this experiment, we provide module data for robot modules manufactured by RobCo[10] – for the remainder of this section, we refer to modules from this set that contain a joint as *joint modules* and static modules as *base, end effector, and links*. Fig. 5 shows the module set used, consisting of four L-shaped links, six I-shaped links and one joint, base, and gripper module each. The objective is to find an assembly that can reach both goals while minimizing the sum of individual module costs. This can be a weighted sum of the acquisition cost, availability preferences, or module mass. For this experiment, we assign costs directly proportional to the module mass. We asked eight experts[11] to find an optimal solution to the task described above within 30 minutes (excluding briefing and debriefing). After each guess, a user interface provides feedback on the cost and goal reachability of the chosen solution. Furthermore, an animation is made available to the expert showing the assembly in the closest configuration to the goals found using a numerical inverse kinematics solver integrated in Timor.

We compare the human performance against a constrained search algorithm and a genetic algorithm (GA). The number of configurations achievable from a given set of modules can be limited through the utilization of Timor's pruning iterators; for the constrained search, robots can have at most five degrees of freedom, at most one successive link between any two joint modules, the first module after the base must contain a joint, and there can be at most one link module between the last joint and the end effector. Under these assumptions, there are

$$\sum_{n=1}^{dof} |J| \cdot ((|L|+1) \cdot |J|)^n = \sum_{n=1}^{5} 11^n = 177,155$$

assemblies with exactly one base and end effector. We integrate the built-in filters that prevent further assembly evaluation if a lighter solution has already been found or if, even without considering the obstacles, no inverse kinematics solution can be found that satisfies joint torque constraints. By leveraging a comprehensive evaluation pipeline within a single library, we are able to pre-filter a large number of

[10]https://www.robco.de/en

[11]Either people with an engineering degree focused on robotics or master students within the last year of a robotics-specific master program. All experts had prior experience working with MRRs.

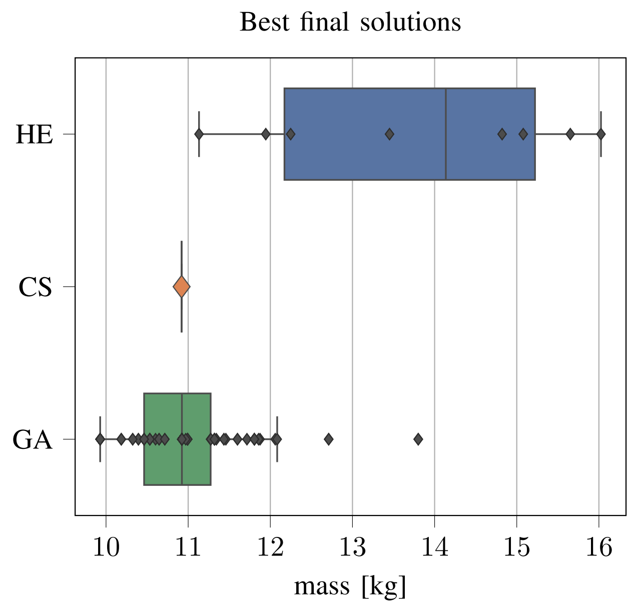


Fig. 6. User study results: The best result (9.92kg) was found using a GA. While the median GA result (10.92kg) is equal to the constrained search (CS) result, none of the human experts (HE) was able to find a solution with a mass of less than 11kg (median: 14.1kg).

solutions prior to model generation. This drastically increases efficiency, leading to a total runtime of 5.5 minutes.

Lastly, we utilize the Timor optimization interface and a genetic algorithm provided by the PyGAD library [39]. Each solution candidate is encoded as a chromosome consisting of thirteen genes that define the module configuration of the robot. We limit the search space to alternating joint and link genes, where each gene can either represent a {link, joint} module (fig. 5) or an empty slot. While restricting the search space to robots with at most six degrees of freedom, other than in the constrained search, an alternation of joints and links is not enforced due to the possibility of empty slots. The solution space consists of

$$(|J|+1)^6 \cdot (|L|+1)^7 = 2^6 \cdot 11^7 = 1,247,178,944$$

different chromosomes – some of which represent equivalent assemblies[12]. We perform 100 trials of 500 generations, each with an average runtime of 21.9 minutes per trial. Fig. 4 shows the final solution found by the best trial.

One of the human experts found a robot with a total mass of 11.13kg, whereas the median expert designed a valid solution with a robot mass of 14.1kg (Fig. 6). Within a fraction of the time, the constrained search yielded a robot with a mass of 10.92kg. Leveraging the goal tolerances, the best trial of the GA resulted in a three degrees-of-freedom robot with a total mass of 9.92kg, while even the worst trial ended with a solution with a total robot mass of 13.8kg (Fig. 6). The best configuration identified by the GA clearly is untypical (Fig. 4): Three joints close to the base in combination with two static, L-shaped links provide enough flexibility for maneuvering between the goals while minimizing the total mass.

[12]The two chromosomes 'J-L1-0-L1-0-0-J1' and 'J-L1-0-0-0-L1-J' are different encodings for a chain of modules with IDs 'J-L1-L1-J'.

V. Conclusion

We introduced and showcased Timor, the first Python toolbox to model, simulate, and optimize modular reconfigurable robots. Our open-source library is published under the MIT license and based on readily available components, offering easy integration in any pipeline without the need for commercial software. The code examples and introductory tutorials are provided with the toolbox repository, along with the source code for all experiments conducted. Bridging the simulation gap between conventional and modular robots, Timor offers features necessary for configuration optimization and model generation of MRRs.

Acknowledgment

The authors gratefully acknowledge financial support by the Horizon 2020 EU Framework Project CONCERT under grant 101016007 and by the ZiM project on energy- and wear-efficient trajectory generation (ZF4086011PO8).

References

[1] M. Yim *et al.*, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[2] J. Liu, X. Zhang, and G. Hao, "Survey on research and development of reconfigurable modular robots," *Advances in Mechanical Engineering*, vol. 8, no. 8, 2016.

[3] C. J. J. Paredis and P. K. Khosla, "Synthesis methodology for task based reconfiguration of modular manipulator systems," in *Proc. of the Int. Symp. on Robotics Research (ISRR)*, 1993.

[4] G. Yang and I.-M. Chen, "1 – introduction," in *Modular Robots: Theory and Practice*, R. S. Han Ding, Ed., 2022, vol. 1, pp. 1–12.

[5] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Self organizing robots based on cell structures – CEBOT," in *Proc. of the Int. Workshop on Intelligent Robots (IROS)*, 1988, pp. 145–150.

[6] D. Schmitz, P. Khosla, and T. Kanade, "The CMU reconfigurable modular manipulator system," Tech. Rep. 88–7, 1988.

[7] A. Yun, D. Moon, J. Ha, S. Kang, and W. Lee, "ModMan: An advanced reconfigurable manipulator system with genderless connector and automatic kinematic modeling algorithm," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4225–4232, 2020.

[8] E. Romiti *et al.*, "Toward a plug-and-work reconfigurable cobot," *Transactions on Mechatronics*, vol. 27, no. 5, pp. 2319–3231, 2022.

[9] 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.

[10] G. C. I-Ming Chen, Song Huat Yeo and G. Yang, "Kernel for modular robot applications: Automatic modeling techniques," *The International Journal of Robotics Research*, vol. 18, no. 2, pp. 225–242, 1999.

[11] M. Mayer, J. Külz, and M. Althoff, "CoBRA: A composable benchmark for robotics applications," arXiv:2203.09337 [cs.RO], 2022.

[12] M. Bordignon, U. Schultz, and K. Stoy, "Model-based kinematics generation for modular mechatronic toolkits," in *Proc. of the ACM SIGPLAN int. Conf. on Generative Programming and Component Engineering (GPCE)*, vol. 9, 2010, pp. 157–166.

[13] M. Bordignon, K. Stoy, and U. Pagh Schultz, "Generalized programming of modular robots through kinematic configurations," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2011, pp. 3659–3666.

[14] M. Feder, A. Giusti, and R. Vidoni, "An approach for automatic generation of the URDF file of modular robots from modules designed using solidworks," *Procedia Computer Science*, vol. 200, pp. 858–864, 2022.

[15] C. Nainer, M. Feder, and A. Giusti, "Automatic generation of kinematics and dynamics model descriptions for modular reconfigurable robot manipulators," in *IEEE Int. Conf. on Automation Science and Engineering (CASE)*, vol. 17, 2021, pp. 45–52.

[16] I.-M. Chen and J. W. Burdick, “Enumerating the non-isomorphic assembly configurations of modular robotic systems,” *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 702–719, 1998.

[17] E. Romiti, N. Kashiri, J. Malzahn, and N. Tsagarakis, “Minimum-effort task-based design optimization of modular reconfigurable robots,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 9891–9897.

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

[19] K. Sims, “Evolving virtual creatures,” in *Proc. of the Ann. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, vol. 21, 1994, pp. 15–22.

[20] D. E. Goldberg and J. H. Holland, “Genetic algorithms and machine learning,” *Machine Learning*, vol. 3, pp. 95–99, 1988.

[21] H. Lipson and J. B. Pollack, “Automatic design and manufacture of robotic lifeforms,” *Nature*, vol. 406, pp. 974–978, 2000.

[22] E. Icer, H. Hassan, K. El-Ayat, and M. Althoff, “Evolutionary cost-optimal composition synthesis of modular robots considering a given task,” in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2017, pp. 3562–3568.

[23] T. Wang, Y. Zhou, S. Fidler, and J. Ba, “Neural graph evolution: Automatic robot design,” in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2019.

[24] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, “Modular robot design synthesis with deep reinforcement learning,” in *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*, vol. 34, no. 06, 2020, pp. 10 418–10 425.

[25] A. Zhao *et al.*, “RoboGrammar: Graph grammar for terrain-optimized robot design,” *ACM Transactions on Graphics*, vol. 39, no. 6, pp. 1–16, 2020.

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

[27] H. Choi *et al.*, “On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward,” *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, 2021.

[28] E. Coumans and Y. Bai, “Pybullet, a Python module for physics simulation for games, robotics and machine learning,” http://pybullet.org, 2021.

[29] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.

[30] J. Lee *et al.*, “DART: Dynamic animation and robotics toolkit,” *Journal of Open Source Software*, vol. 3, no. 22, 2018.

[31] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149–2154.

[32] P. Corke and J. Haviland, “Not your grandmother’s toolbox – the robotics toolbox reinvented for Python,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2021, pp. 11 357–11 363.

[33] J. Carpentier *et al.*, “The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *Proc. of the IEEE/SICE Int. Symp. on System Integration (SII)*, 2019, pp. 614–619.

[34] D. Christensen, D. Brandt, K. Stoy, and U. Schultz, “A unified simulator for self-reconfigurable robots,” in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008, pp. 870–876.

[35] T. Collins and W.-M. Shen, “ReBots: A drag-and-drop high-performance simulator for modular and self-reconfigurable robots,” University of Southern California, Tech. Rep. 714, 2016.

[36] P. Thalamy, B. Piranda, A. Naz, and J. Bourgeois, “VisibleSim: A behavioral simulation framework for lattice modular robots,” *Robotics and Automation Systems*, vol. 147, no. C, 2022.

[37] J. Carpentier, F. Valenza, N. Mansard *et al.*, “Pinocchio: fast forward and inverse dynamics for poly-articulated systems,” https://stack-of-tasks.github.io/pinocchio, 2015.

[38] J. Pan, S. Chitta, and D. Manocha, “FCL: A general purpose library for collision and proximity queries,” in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 3859–3866.

[39] A. F. Gad, “PyGAD: An intuitive genetic algorithm Python library,” arXiv:2106.06158 [cs.NE], 2021.