2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

CoBRA: A Composable Benchmark for Robotics Applications

Matthias Mayer, Jonathan Külz, and Matthias Althoff

***Abstract*— Selecting an optimal robot, its base pose, and trajectory for a given task is currently mainly done by human expertise or trial and error. To evaluate automatic approaches to this combined optimization problem, we introduce a benchmark suite encompassing a unified format for robots, environments, and task descriptions. Our benchmark suite is especially useful for modular robots, where the multitude of robots that can be assembled creates a host of additional parameters to optimize. We include tasks such as machine tending and welding in synthetic environments and 3D scans of real-world machine shops. All benchmarks are accessible through cobra.cps.cit.tum.de, a platform to conveniently share, reference, and compare tasks, robot models, and solutions.**

I. Introduction

Scientific research benefits when results are reproducible and easily comparable to alternative solutions. For instance, in computer science and robotics, computer vision benchmarks like ImageNet [1] or MS-COCO [2] have brought tremendous progress. One key feature is that they broke down visual perception into tasks of varying difficulty, from single, cropped frame labeling to detecting multiple objects. These benchmarks certainly coincided with the resurgence of (deep) learning and possibly enabled it in the first place [2]. An area in robotics where multiple benchmarks exist is grasping and/or bin picking [3]–[5]; more are discussed in [6, Tab. 1]. Especially Dex-Net [5] co-develops both novel solutions for grasp planning and improving them through publishing data sets for training and evaluation.

Within the motion planning community, only a few benchmarks are established, e.g., by the creators of the Open Motion Planning Library (OMPL) [7], [8][1], or Parasol[2]. These are either limited to simple point-to-point planning or only contain abstract planning problems without a specific application. In contrast, a benchmark suite specialized in a specific use case is CommonRoad for autonomous driving [9] or MotionBenchMaker for manipulator motion planning [6]. However, no benchmark suite exists for evaluating optimal robots or modular robot assemblies for a given task. We provide the first benchmark suite to compare robots and modular robot assemblies in different real-world environments for various cost functions. An example solution to a benchmark task defined in a 3D-scanned environment is shown in Fig. 1.

All authors are with the Technical University of Munich, TUM School of Computation, Information and Technology, Chair of Robotics, Artificial Intelligence and Real-time Systems, Boltzmannstraße 3, 85748, Garching, Germany. {matthias.mayer, jonathan.kuelz, althoff}@tum.de

[1]plannerarena.org/, accessed on Feb. 29th, 2024

[2]parasol.tamu.edu/groups/amatogroup/benchmarks/, accessed on Feb. 29th, 2024

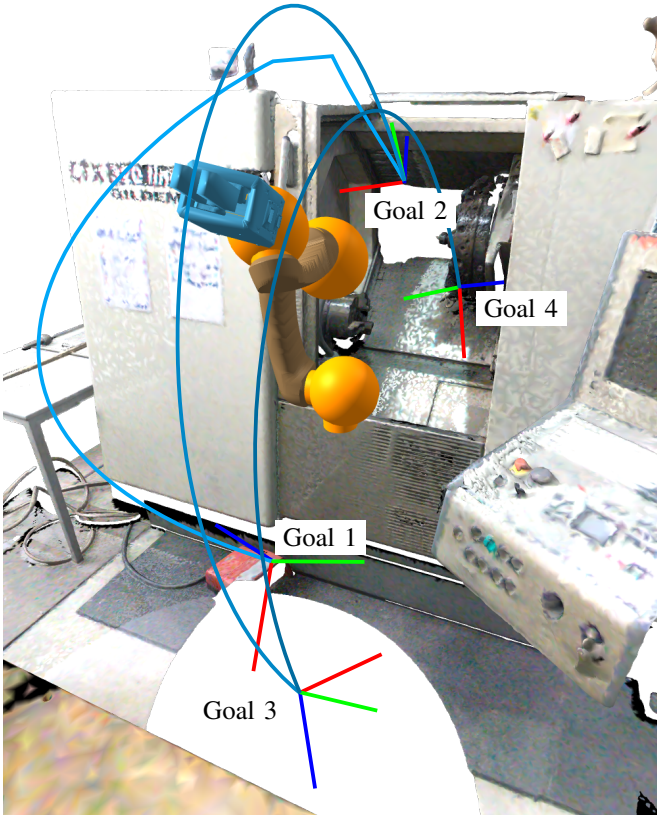

Fig. 1. A robot solving the machine tending task Liu2020/Case2b/Schunk_IWB_CTX_300_linear_0 minimizing mechanical energy. The robot is assembled with the module set $\mathcal{R}$ = IMPROV in the order $M = [1, 21, 4, 22, 5, 23, 12]$ and its end-effector follows the cyan path $\mathbf{T}_{\text{eef}}$. An animation is available at cobra.cps.cit.tum.de/ICRA24/example.

In our literature overview, we survey robot descriptions, common robotic tasks in industry, and modular robot configuration optimization; we also state the objectives of our benchmark suite.

A. Related Work

1) Robot (Task) Description: An extensive and continuously updated overview of domain-specific languages for robotics is provided in [10]. However, we have not found a common framework to describe (modular) robots, tasks, and cost functions. Nonetheless, [11] inspired the extension of our previous module description [12] detailed in Sec. III-A.

MoveIt! [13] is a common software stack for robotic motion planning, which integrates OMPL [14] for planning within the Robot Operating System (ROS) [15]. On top of MoveIt, the work in [16] creates solutions to many robotic tasks

TABLE I

COMMON ROBOT TASKS IN INDUSTRY WITH THEIR MARKET SHARE AND PREDOMINANT ACTIONS [17, P. 14].

Task	Market Share	Predominant Actions
Handling / Tending	48.1 %	PTP, Trajectory
Welding	15.7 %	PTP, Trajectory
Assembly	11.0 %	PTP, Trajectory, Force Control
Dispensing	5.1 %	Trajectory
Predominant Processing	1.1 %	Trajectory, Force Control
Other & Unspecified	19.0 %	(Indeterminable)

with interdependent sub-tasks. Due to the deep integration in MoveIt, their task description is not portable.

2) Typical Robotic Tasks: Our **Co**mposable **B**enchmark (suite) for **R**obotics **A**pplications (CoBRA) intends to capture the variety of real-world applications a robot might encounter. Based on an analysis of market shares of different applications in robotics from 2023 [17], these are still mainly in manufacturing (see Tab. I). In [18], [19, Ch. 54.4] we found descriptions of the type of actions mainly required for each of these tasks. Most tasks can be broken down into point-to-point movements (PTP), (fixed) Cartesian trajectories, or force control. Some of these processes may need additional feedback, e.g., from vision, which is needed for grasp planning, quality assessment, and reworking.

3) Optimizing Modular Robots: One key aspect we address with our benchmark suite is modular and/or reconfigurable robotics [19, Ch. 22.2], where even small module sets can be used to assemble millions of possible robots [20]. Recent solutions to find optimal assemblies have combined hierarchical elimination with kinematic restrictions [12], [20], genetic algorithms [21], heuristic search [22], and reinforcement learning [23]. The above-mentioned methods use modules and create modular robots with significantly different properties. Comparisons of the above-mentioned optimization methods are either not performed at all (e.g., [12], [22]) or use re-implementations of other work and thereby impair comparability [23]. Analyzing these optimizers on a shared set of tasks and robots enables fair comparisons.

B. Contributions

We introduce CoBRA, a composable benchmark (suite) for robotics applications, which tests the selection, synthesis, and programming of robots based on well-defined environments, (modular) robot models, and task descriptions. Perception of and reaction to an unexpected environment are intentionally left out for now to focus on evaluating robot assemblies for the intended industrial tasks with known and controlled environments.

Our benchmark suite strives for the same properties as [9]:

- **Unambiguous**: The entire benchmark settings can be referred to by a unique identifier, and all details are provided in manuals on our website.
- **Composable**: Splitting each benchmark into components (robot model, tasks, and cost functions) allows one to easily compose new benchmarks by recombining existing components.
- **Representative**: Our benchmark suite contains real 3D environments and hand-crafted tasks covering various robotic tasks identified in the literature.
- **Portable**: All components of our benchmark suite are described in the JSON format, which makes it independent of specific platforms and programming languages. An interface to URDF eases the integration of the robots into different workflows.
- **Scalable**: Tasks can describe simple pick-and-place tasks as well as complex tasks, such as polishing a surface. Our robot description is valid for standard serial kinematics and extendable to modular robots with (multiple) closed kinematic chains and complex modules with multiple bodies, joints, connection points, bases, and end effectors.
- **Open**: Our benchmark suite is published in an open format on our website free of charge. Benchmark suggestions from the community are welcome.
- **Independent**: The benchmark descriptions are independent of any tools or products, making them suitable for any robot design and task.

The remaining paper is organized as follows: In Sec. II, we provide the formal definitions of the optimization problem to be solved in each benchmark. Next, in Sec. III, we describe the implementation of the robots, cost functions, and task descriptions. Lastly, in Sec. IV, we state our current approach for task generation and provide an example.

II. TASK DESCRIPTION AND PROBLEM STATEMENT

Every benchmark B comprises a set of robot modules $\mathcal{R}$ (possibly a single robot), a cost function with ID C, and a task Θ. Each task itself includes obstacles $\mathcal{W}_{occ}$, a set of constraints $\mathcal{C}$, and a set of goals $\mathcal{G}$.

Within a given release of the benchmark suite, a benchmark B can be written as

$$B = \mathcal{R} : C : \Theta. \tag{1}$$

An example is $\mathcal{R}$ = Panda, C = cyc (cycle time), Θ = Factory1 resulting in the benchmark ID `Panda:cyc:Factory1`. As in CommonRoad [9], one can integrate individual or novel module sets, cost functions, or tasks and denote them by the keyword *IND* or indicate modification by prefixing them with *M-*. Both *IND* and *M-* need auxiliary explanations in the accompanying publications. Collaborative robot tasks can also be described, where $n \in \mathbb{N}^+$ robots work in a common task Θ and each robot fulfills tasks with its individual cost function C_1 to C_n:

$$B = [\mathcal{R}_1, ..., \mathcal{R}_n] : [C_1, ..., C_n] : C\text{-}\Theta. \tag{2}$$

Note that the prefix *C-* of task Θ indicates that it is collaborative. The prefix *M-* precedes *C-*.

TABLE II

AVAILABLE PROJECTIONS TO COORDINATES FROM [19, TAB. 1.2].

	Projection Name	Coordinates
Translation $\mathbf{t}(\mathbf{T})$	Cartesian	$[x, y, z]$
	Cylindrical	$[r_{cyl}, \theta, z]$
	Spherical	$[r_{sph}, \theta, \phi]$
Rotation $\mathbf{R}(\mathbf{T})$	XYZ fixed	$[r, p, y]$
	ZYX Euler	$[\alpha, \beta, \gamma]$
	Axis-angle	$[n_x, n_y, n_z, \theta_R]$
	Quaternion	$[a, b, c, d]$

A. Poses

Our benchmarks use poses $\mathbf{T} \in \mathbb{SE}(3)$[3]. We denote the rotation matrix $\mathbf{R}$ in $\mathbf{T}$ as $\mathbf{R}(\mathbf{T}) \in \mathbb{SO}(3)^3$ and the translation $\mathbf{t}$ in $\mathbf{T}$ as $\mathbf{t}(\mathbf{T}) \in \mathbb{R}^3$. With these, any point represented as $\mathbf{p}_a \in \mathbb{R}^3$ in frame a is represented with respect to frame b by:

$$\mathbf{p}_b = \mathbf{T}_b^a \mathbf{p}_a = \mathbf{R}(\mathbf{T}_b^a)\mathbf{p}_a + \mathbf{t}(\mathbf{T}_b^a)\ ^4. \tag{3}$$

To constrain and judge the distance between poses, we use a notation similar to [24, Sec. IV.A]: For subsequent formalizations, we introduce the mapping $S\colon \mathbb{SE}(3) \mapsto \mathbb{R}^n, n \in \mathbb{N}^+$ and the shorthand $\Delta_S(\mathbf{T}, \mathbf{T}_\mathrm{d}) = S(\mathbf{T}_\mathrm{d}^{-1}\mathbf{T})$ to denote the difference between a pose $\mathbf{T}$ and a desired pose $\mathbf{T}_\mathrm{d} \in \mathbb{SE}(3)$ after applying S. S can be any combination of the projections listed in Tab. II, e.g., the default mapping $S(\mathbf{T}) = [x(\mathbf{T}), y(\mathbf{T}), z(\mathbf{T}), \theta_R(\mathbf{T})]^T$ contains the three Cartesian coordinates and the rotation angle about an axis for any pose $\mathbf{T}$.

Most real-world tasks accept some tolerance in execution so that each coordinate i can vary between a minimum $\gamma_i^{\max} \in \mathbb{R}$ and maximum value $\gamma_i^{\max} \in \mathbb{R}$, respectively, which we denote by the interval vector $\Gamma(\mathbf{T}_\mathrm{d}) = [\gamma^{\min}, \gamma^{\max}] \in \mathbb{R}^{2\times N}$. We say $\mathbf{T}$ fulfills $\mathbf{T}_\mathrm{d}$ if $\Delta_S(\mathbf{T}, \mathbf{T}_\mathrm{d}) \in \Gamma(\mathbf{T}_\mathrm{d})$. As an example, one may want to constrain the end effector to be upright (z-axis pointing upwards), which can be done by using a desired pose $\mathbf{T}_\mathrm{d} = \mathbf{I}_{4\times4}$ (four-by-four identity matrix) in the world frame, the projections $S(\mathbf{T}) = [r(\mathbf{T}), p(\mathbf{T})]^T$, and setting both intervals from $\gamma^{\min} = -10^{-3}$ to $\gamma^{\max} = 10^{-3}$.

B. Hybrid Motion Planning Problem

Our benchmark suite extends classical motion planning for robotics [25], which only searches for a motion $\mathbf{z}(t)$ over time t satisfying a set of goals $\mathcal{G}$ and constraints $\mathcal{C}$, detailed later on. We extend this planning problem by

- synthesizing robots from modules, resulting in a tuple of n modules $M = [m_1, ..., m_n]$, $m_i \in \mathcal{R}$ to be assembled for serial kinematics;[5]
- optimizing the base pose $\mathbf{B} \in \mathbb{SE}(3)$ (see Sec. II-A).

[3]$\mathbb{SO}(3)$ is the special orthogonal group represented as 3×3 rotation matrices; $\mathbb{SE}(3) = \mathbb{R}^3 \times \mathbb{SO}(3)$ represents both translations and rotations.

[4]We use points $\mathbf{p} = [p_x, p_y, p_z]^T$ and their homogeneous extension $\mathbf{p} = [p_x, p_y, p_z, 1]^T$ interchangeably so that $\mathbf{p}_b = \mathbf{T}_b^a\mathbf{p}_a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_a \\ 1 \end{bmatrix}$.

[5]Extensions to parallel kinematics are discussed in cobra.cps.cit.tum.de →Documentation →Robot.

For a (rigid) robot with $n \in \mathbb{N}^+$ joints at time t, the state of the robot is given by the joint configuration $\mathbf{q}(t) \in \mathbb{R}^n$ and its derivative $\dot{\mathbf{q}}(t)$. The state changes by the commanded accelerations $\ddot{\mathbf{q}}(t)$. We combine both in the state and input vector $\mathbf{z}(t) = (\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t))$. If unnecessary, we omit the dependency on time. Based on the information provided in the corresponding benchmark, we can construct a robot model, including its kinematics and dynamics [12]. This model includes functions to calculate the

- end effector pose $\mathbf{T}_\mathrm{eef}(\mathbf{q}, M) \in \mathbb{SE}(3)$ relative to the base $\mathbf{B}$ [12]; for inverse solutions we use [19, Sec. 2.7];
- robot occupancy $\mathcal{O}(\mathbf{q}, \mathbf{B}, M) \subset \mathcal{P}(\mathbb{R}^3)$, where $\mathcal{P}(\bullet)$ returns the power set of $\bullet$;
- forward dynamics given control forces $\boldsymbol{\tau}$ and the external wrench $\mathbf{f}_\mathrm{ext}(t)$: $\ddot{\mathbf{q}} = \mathrm{dyn}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}, \mathbf{f}_\mathrm{ext}, \mathbf{B}, M)$ [12];
- inv. dynamics $\boldsymbol{\tau} = \mathrm{dyn}^{-1}(\mathbf{z}, \mathbf{f}_\mathrm{ext}, \mathbf{B}, M)$ [19, Ch. 3.5].

The functions defined above allow us to define cost functions J_C for each cost ID C. These evaluate a proposed solution with its base pose $\mathbf{B}$, module order M, and a motion specified for the time interval $[0, t_\mathrm{N}]$ (without loss of generality, we set $t_0 = 0$). For J_C we add terminal costs Φ_C and the integral of running costs L_C:

$$J_C(\mathbf{z}(t), t_\mathrm{N}, \mathbf{B}, M) = \Phi_C(\mathbf{z}(0), \mathbf{z}(t_\mathrm{N}), t_\mathrm{N}, \mathbf{B}, M) + \int_0^{t_\mathrm{N}} L_C(\mathbf{z}(t'), t', \mathbf{B}, M)\mathrm{d}t'. \tag{4}$$

Formally, we define the *hybrid motion planning problem* as finding a module order M^*, base placement $\mathbf{B}^*$, and motion vector $\mathbf{z}^*$ minimizing a given cost function J_C:

$$[M^*, \mathbf{B}^*, \mathbf{z}^*] = \underset{M, \mathbf{B}, \mathbf{z}}{\arg\min}\, J_C(\mathbf{z}(t), t_\mathrm{N}, \mathbf{B}, M) \tag{5}$$

subject to $\forall t \in [0, t_\mathrm{N}]$:

$$\ddot{\mathbf{q}} = \mathrm{dyn}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}, \mathbf{f}_\mathrm{ext}, \mathbf{B}, M) \tag{6}$$

$$\forall c \in \mathcal{C}\colon c(\mathbf{z}, t, \mathbf{B}, M) \leq 0. \tag{7}$$

Initially, the robot must satisfy all constraints in $\mathcal{C}$ (see Sec. II-C) and be stationary, i.e., $\mathbf{z}(0) = [\mathbf{q}(0), \mathbf{0}_n, \mathbf{0}_n]$, where $\mathbf{0}_n$ is a vector of n zeros. Note that if the available module set $\mathcal{R}$ only includes one valid robot, and $\mathbf{B}$ is restricted to a single pose, this hybrid motion planning problem is reduced to standard motion planning for robotic manipulators. Subsequently, we will introduce our definitions for constraints in $\mathcal{C}$ and goals in $\mathcal{G}$.

C. Constraints

We consider constraints that can be written as $c(\mathbf{z}(t), t, \mathbf{B}, M) \leq 0$, noting that equality can be ensured by adding the partial constraints $c_i \leq 0 \wedge -c_i \leq 0$. To incorporate Boolean functions b in our constraint formulation, we use the operator $\mathbb{I}(b)$, which evaluates to -1 if b is true and otherwise to 1. We split $\mathcal{C}$ into constraints holding for an entire task Θ, named $\mathcal{C}_S$, and those applying to a single goal $G_i \in \mathcal{G}$ named $\mathcal{C}_i$. Additionally, we introduce the

- robot Jacobian $J(\mathbf{q}) = \frac{\mathrm{d}\mathbf{T}_\mathrm{eef}(\mathbf{q})}{\mathrm{d}\mathbf{q}}$;

TABLE III

AVAILABLE CONSTRAINT FUNCTIONS.

Constraint	Constraint Function
Joint position limits	$\mathbf{q} \geq \mathbf{q}_{\min} \wedge \mathbf{q} \leq \mathbf{q}_{\max}$
Joint velocity and acceleration limits	$\lvert\dot{\mathbf{q}}\rvert \leq \dot{\mathbf{q}}_{\max}, \lvert\ddot{\mathbf{q}}\rvert \leq \ddot{\mathbf{q}}_{\max}$
Joint torque limits	$\lvert\mathrm{dyn}^{-1}(\mathbf{z}, \mathbf{f}_{\text{ext}}, \mathbf{B}, M)\rvert \leq \boldsymbol{\tau}_{\max}$
End effector velocity	$\lVert\mathbf{v}\rVert_2 \in [v_{\min}, v_{\max}]$, $\lVert\boldsymbol{\omega}\rVert_2 \in [\omega_{\min}, \omega_{\max}]$
Keep $\mathbf{T}_{\text{eef}}$ close to $\mathbf{T}_{\text{d}}$	$\Delta_S(\mathbf{T}_{\text{eef}}, \mathbf{T}_{\text{d}}) \in \Gamma(\mathbf{T}_{\text{d}})$
Fulfill all goals	$\forall G_i \in \mathcal{G}\, \exists t_{\text{f},i} \in [0, t_{\text{N}}]$
Fulfill all goals in order	$\forall G_i, G_j \in \mathcal{G}, i < j \colon t_{\text{f},i} < t_{\text{f},j}$
No self-collision	$\forall l, l' \in \mathcal{L},\, l \neq l' \colon \tilde{\mathcal{O}}(l) \cap \tilde{\mathcal{O}}(l') = \emptyset$
No collision	$\mathcal{O}(\mathbf{q}, \mathbf{B}, M) \cap \mathcal{W}_{occ} = \emptyset$
Valid base pose $\mathbf{B}_{\text{given}}$	$\Delta_S(\mathbf{B}, \mathbf{B}_{\text{given}}) \in \Gamma(\mathbf{B}_{\text{given}})$
Valid assembly	see Sec. III-A

- velocities of the end effector $[\mathbf{v}, \boldsymbol{\omega}] = J(\mathbf{q})\dot{\mathbf{q}}$;
- occupancy of link l from set of robot links $\mathcal{L}$ as $\tilde{\mathcal{O}}(l)$;
- desired base pose with tolerance $\mathbf{B}_{\text{given}}$.

For vectors $\mathbf{v} \in \mathbb{R}^n$, we denote the componentwise absolute value as $|\mathbf{v}|$ and the 2-norm as $\|\mathbf{v}\|_2$. The time to finish a single goal G_i is $t_{\text{f},i}$. With these, we list the supported constraints to formalize $\mathcal{C}_S$ and $\mathcal{C}_i$ in Tab. III.

D. Goals

In every task Θ, all robots have to fulfill a set of n goals $\mathcal{G} = \{G_1, ..., G_n\}$. As some goals are not fulfilled instantaneous, we introduce the duration operator $t_{\text{d}}(\bullet)$ which returns the length of $\bullet$, such as a goal or trajectory, in seconds. Each goal $G_i \in \mathcal{G}$ contains a set of constraint functions $\mathcal{C}_i$, a termination condition $g_i(\mathbf{z}, t, \mathbf{B}, M)$, and an execution duration $t_{\text{d}}(G_i)$. For convenience, we order the goals such that G_i is fulfilled after G_{i-1}. A goal G_i is fulfilled at time $t_{\text{f},i}$ if $g_i(\mathbf{z}, t, \mathbf{B}, M)$ evaluates to true and its constraints have been satisfied within its duration $t_{\text{d}}(G_i)$: $\forall t' \in [t_{\text{f},i} - t_{\text{d}}(G_i), t_{\text{f},i}], \forall c \in \mathcal{C}_i \colon c(\mathbf{z}(t'), t', \mathbf{B}, M) \leq 0$.

We introduce a small deviation $\epsilon = 10^{-3}$ and a Cartesian trajectory $\mathbf{T}(t)$ describing desired poses for each time step within $[0, t_{\text{d}}(\mathbf{T}(t))]$. Some goals may have an explicit duration, such as a pause for t_{d} seconds. Additionally, $\mathcal{W}_A \subset \mathcal{P}(\mathbb{R}^3)$ is a space to temporarily leave, e.g., the workspace of a machine (see Sec. III-C). With these prerequisites, we expect that termination conditions g for most tasks given in Tab. I can be composed of the predicates in Tab. IV.

With respect to Tab. I, *at* and *reach* model simple PTP tasks. The predicate *followed* allows one to define more complex trajectories to be followed with precise timing. *ReturnTo*, *pause*, and *left* specify synchronization with external events, such as a CNC mill processing a placed part, without restricting the robot to specific positions.

TABLE IV

AVAILABLE GOAL PREDICATES.

Goal	Goal Predicate
$\mathrm{at}(\mathbf{T}_{\text{d}}, \mathbf{q})$	$\Delta_S(\mathbf{B}\,\mathbf{T}_{\text{eef}}(\mathbf{q}, M), \mathbf{T}_{\text{d}}) \in \Gamma(\mathbf{T}_{\text{d}})$
$\mathrm{reach}(\mathbf{T}_{\text{d}}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$	$\mathrm{at}(\mathbf{T}_{\text{d}}, \mathbf{q}) \wedge \lVert\dot{\mathbf{q}}\rVert_2 \leq \epsilon \wedge \lVert\ddot{\mathbf{q}}\rVert_2 \leq \epsilon$
$\mathrm{followed}(\mathbf{T}(t), \mathbf{q}, t)$	$\forall t' \in [0, t_{\text{d}}(\mathbf{T}(t))]$: $\mathrm{at}(\mathbf{T}(t_{\text{d}}(\mathbf{T}(t)) - t'), \mathbf{q}(t - t'))$
$\mathrm{returnTo}(G_i, t)$	$\lVert\mathbf{q}(t) - \mathbf{q}(t_{\text{f},i})\rVert_2 \leq \epsilon \wedge \lVert\dot{\mathbf{q}}(t) - \dot{\mathbf{q}}(t_{\text{f},i})\rVert_2 \leq \epsilon \wedge \lVert\ddot{\mathbf{q}}(t) - \ddot{\mathbf{q}}(t_{\text{f},i})\rVert_2 \leq \epsilon$
$\mathrm{pause}(\mathbf{q}, t_{\text{d}}, t)$	$\forall t' \in [t - t_{\text{d}}, t] \colon \lVert\mathbf{q}(t') - \mathbf{q}(t)\rVert_2 \leq \epsilon$
$\mathrm{left}(\mathcal{W}_A, \mathbf{q}, t_{\text{d}}, t)$	$\forall t' \in [t - t_{\text{d}}, t] \colon \mathcal{O}(\mathbf{q}(t'), \mathbf{B}, M) \cap \mathcal{W}_A = \emptyset$

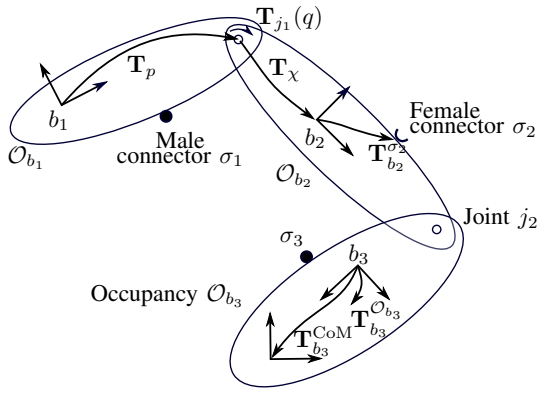

Fig. 2. Sketch of a module consisting of three ellipsoidal bodies b_1, b_2, b_3, each with a body-fixed frame. The bodies are connected via two joints j_1, j_2 (empty circles), and the joint transformation $\mathbf{T}_p\mathbf{T}_{j_1}(\mathbf{q})\mathbf{T}_\chi$ is shown between b_1 and b_2. Each body has a connector $\sigma_1, \sigma_2, \sigma_3$, and the pose $\mathbf{T}_{b_2}^{\sigma_2}$ for connector σ_2 is shown. Body b_3 displays the transformation $\mathbf{T}_{b_3}^{\text{CoM}}$ to its center of mass and $\mathbf{T}_{b_3}^{\mathcal{O}_{b_3}}$ to the ellipse describing its occupancy $\mathcal{O}_{b_3}$.

III. IMPLEMENTATION

The next three subsections provide implementation details for modeling robots, define cost functions, and describe how tasks and proposed solutions are stored.

A. Robots

We propose a model description similar to the universal robot description format (URDF)[6] and alter it such that *modules* m_i, rather than assembled robots, are the basic entities. Modules generalize our previous description from [12], considering more than two rigid bodies $b_{1,2}$ and a single joint j_1. They also integrate *connectors* σ, as introduced by [11], to explicitly model valid assemblies.

Each module m_i has a unique $\texttt{ID}(m_i)$ within a module set $\mathcal{R}$ with a unique $\texttt{name}(\mathcal{R})$. Any robot with module order M can, therefore, be described as a string $\texttt{name}(\mathcal{R})\texttt{.[ID}(m_1, \ldots, m_N)\texttt{]}$, see Sec. IV. CoBRA provides a web API to generate URDFs for any included robot[7].

[6]wiki.ros.org/urdf/XML, accessed on Feb. 29th, 2024

[7]For example cobra.cps.cit.tum.de/ICRA24/urdf provides the URDF for the robot in Fig. 1.

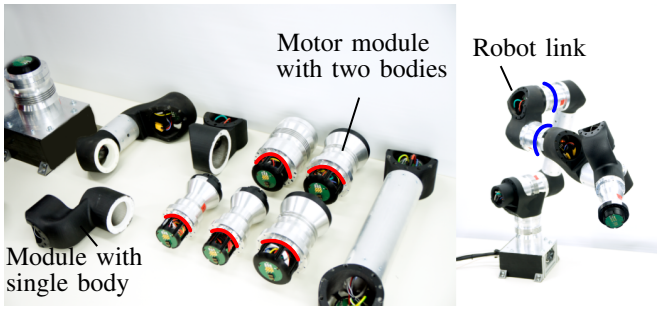

Fig. 3. Modules (left) and links in the assembled robot (right) from the module set PROMODULAR with red lines marking the separation of bodies in a module and blue lines separating a link in the assembled robot. Based on [20, Fig. 1].

Similar to [11], [12], we compose modules of *bodies* b_i and *joints*. Fig. 2 illustrates a single module, and Fig. 3 is an example of real modules assembled into a robot. Each body b_i specifies the dynamics, e.g., the center of mass *CoM*, and the occupancy $\mathcal{O}_{b_i}$, e.g., each ellipse in Fig. 2, of a rigid body. Additionally, each module details how to connect it to others via *connectors* σ. Each connector contains lists of supported *sizes* $\in \mathcal{P}(\mathbb{R})$ and *types* $\in \mathcal{P}(\texttt{string})$. Multiple sizes realize module designs where adjacent sizes can fit together, but not all combinations are valid; similarly, multiple types enable the construction of modules that fit with multiple connection designs. Note that bodies from multiple modules are connected rigidly and together form a link, as known, e.g., from URDF; the joints within any module separate links. We denote the set of links in the assembled robot as $\mathcal{L}$.

A robot assembly is *valid* if connectors match, i.e., they share at least a common entry in their respective *size* and *type* lists, and their *genders* are both hermaphroditic (gender-less) or opposing (male/female). This matching extends the one based on gender alone, as introduced in [11]. Additionally, each connector σ defines its *pose* relative to the body frame b_i as $\mathbf{T}^{\sigma}_{b_i}$, shown for b_2 in Fig. 2. If two connectors $\sigma_{A,B}$ with pose $\mathbf{T}^{\sigma_A}_{A}, \mathbf{T}^{\sigma_B}_{B}$, relative to body A and B, are connected during assembly, their x-axes align, and their z-axes are anti-parallel. For each valid assembly of modules, we can generate kinematic, dynamic, and collision models[8]. As in [11], an unlimited number of connectors on a module enables us to also model robots with branching or closed kinematic chains[8].

A joint j_i connects two bodies, referred to as *parent* p and *child* χ. Each joint j_i needs three transformations: $\mathbf{T}_p$ from the parent p to the joint frame, the joint transformation $\mathbf{T}_{j_i}(q, \text{type})$ and $\mathbf{T}_\chi$, from the joint to the child χ. The most common *type* of joint is *revolute*, where $\mathbf{T}_{j_i}(q, \text{revolute})$ is a single rotation about the z-axis of the joint frame by the angle q. An example is visualized between body b_1 and b_2 in Fig. 2. In addition, we also model the drivetrain dynamics for joints, with gear ratio k, motor side inertia I_m, Coulomb and viscous friction f_c, f_v, respectively. These result in an additional torque τ_i in joint j_i [26, Ch. 7]

$$\tau_i = I_\text{m} k^2 \ddot{q}_i + f_\text{v} \dot{q}_i + f_\text{c} \operatorname{sign}(\dot{q}_i). \tag{8}$$

Currently, we provide the set of robot modules IMPROV based on Schunk's LWA 4P [12] and PROMODULAR from [20]. Additionally, we provide standard industrial robots, such as Franka Emika's Panda, with dynamic parameters identified in [27] as a point of reference.

```
[1] task
├─ [1] header
│  ├─ [1] taskID: string
│  ├─ [1] version: string
│  ├─ [1..] author: List[string]
│  ├─ [0..] tags: string = []
│  ├─ [1] date: string yyyy-MM-dd
│  └─ [0..1] gravity: ℝ³ = [0, 0, −9.81]ᵀ
├─ [0..] obstacles
├─ [0..] constraints
└─ [1..] goals
   ├─ [1] ID: string
   ├─ [1] type: {at, reach, returnTo, pause,
   │  follow, leave}
   └─ [1] parameter (specific to type)
```

Fig. 4. Abbreviated structure of the task object. For each attribute, we provide ranges for the number of allowed elements. Fields with a lower bound of 0 elements are optional and come with default values. Fields that allow more than one element are arrays. For each primitive field, the type is given after a colon. The complete version, including all fields and drafted extensions, is published on our website[10].

B. Cost Functions

We include cost functions used in (modular) robot optimization from the literature in CoBRA. In general, they can be split into *atomic* functions that map a robot and/or its motion to a scalar value or compound functions that weigh multiple atomic functions. A detailed description of all available costs can be found on our website[9].

In our example in Sec. IV, we use atomic costs describing the robot mass (J_m) in kilograms, number of actuated joints (J_numJ), the time it takes to move the robot (J_T) in seconds and the mechanical energy required (J_mechE) in joules. In addition, we evaluated two combined cost functions: Liu2 weighs mechanical energy and execution time so that $J_\text{Liu2} = J_\text{mechE} + 0.2\, J_\text{T}$ [20, Sec. IV B] and Whit2 the number of actuated joints and the robot mass so that $J_\text{Whit2} = 0.025 J_\text{numJ} + 0.1\, J_\text{m}$ [23, Tab. 1]. Any other weighted sum of N cost functions can be abbreviated as $[(J_1|w_1), ..., (J_N|w_N)]$ to define a total $J_C = \sum_{i=1}^{N} w_i J_i$.

C. Tasks

The structure of a task is shown in Fig. 4, describing its obstacles $\mathcal{W}_{occ}$, constraints $\mathcal{C}$ (Sec. II-C) and goals $\mathcal{G}$ (Sec. II-D). A task ID and a benchmark version uniquely identify each task. Additionally, it contains contact information from the

[8]cobra.cps.cit.tum.de →Documentation →Robot

[9]cobra.cps.cit.tum.de →Documentation →Solution

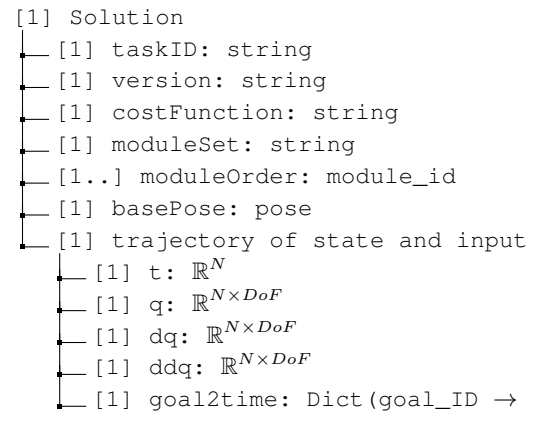

Fig. 5. Abbreviated structure of the solution object for a robot with a single kinematic chain following the notation from Fig. 4. The complete version is published on our website[9].

authors, tags for semantic searches, and a date of publishing. The complete description can be found on our website[10].

Obstacles are stationary parts of the environment, such as machines or columns, that are placed at a pose relative to the world frame and are represented by collision and optional visual geometries. Those two geometries enable efficient collision checking on simpler over-approximations while keeping fidelity for visualization if needed.

For every constraint and goal, the task must reference the specific functions in Sec. II-C, II-D via a type and provide their parameters, such as

- poses (with tolerances) $\mathbf{T}$ (or an array of these to follow)[10]);
- references G_i to other goals via their ID;
- scalars, such as, duration t_d for pause or $v_\mathrm{min}, v_\mathrm{max}$ for end-effector constraints;
- arrays of goal IDs to fulfill in order;
- regions $\mathcal{W}_A$, which can be specified as any geometry, as defined for an obstacle.

The solution structure for a robot with a single kinematic chain is provided in Fig. 5. It specifies the task Θ by its ID and benchmark version, the ID of the used cost function C, and a robot via module set $\mathcal{R}$, module order M^*, and base pose $\mathbf{B}^*$. The solving state and input vector $\mathbf{z}^*$ as defined in (5) is also given. It contains a sequence of N samples of $\mathbf{z} = [\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}]$ at times in `t`. For each goal in a task $G_i \in \mathcal{G}$, `goal2time` states its final time $t_{\mathrm{f},i}$.

Any solution can be submitted to our website where we check whether it solves the specified task Θ while satisfying the constraint set $\mathcal{C}$. Valid solutions will be published together with the provided author information and can be queried, e.g., for used cost functions or final cost.

IV. Numerical Example

Following (1), we compare solutions to benchmarks with

- module sets $\mathcal{R} \in \{\texttt{PROMODULAR}, \texttt{Panda}, \texttt{IMPROV}\}$,
- cost function IDs $C \in \{\mathrm{mechE}, \mathrm{Whit2}, \mathrm{Liu2}\}$, and

[10]cobra.cps.cit.tum.de →Documentation →Task

TABLE V

Minimal solution costs for the example task with different robots.

Cost J_i (from III-B)	`PROMODULAR`	`Panda`	`IMPROV`
J_mechE	1194.6	255.7	383.5
J_Whit2	2.085	1.84	1.522
J_Liu2	1198.8	260.7	386.4

- a single task Θ = `Liu2020/Case2b/Schunk_IWB_CTX_300_linear_0`.

Within Θ, the goals $\mathcal{G}$ require the robot to move in and out of a 3D-scanned CNC machine. We include *joint limits* (in position and torque), *no (self-)collisions*, and a fixed *goal order* as constraints $\mathcal{C}$ (Sec. II-C). An example solution is shown in Fig. 1 and all solutions are on our website[11].

Tab. V summarizes the minimal costs J_i of the generated solutions for a robot from each set $\mathcal{R}$ over ten generated trajectories. All solution trajectories were generated with OMPL's RRT-Connect implementation[12] and adhere to the constraints given in Θ.

V. Conclusions

This paper addressed the problem of finding optimal robotic solutions for industrial tasks using conventional and modular robots. We propose a novel framework that integrates motion planning and modular robot optimization, which together can be evaluated on a set of realistic tasks in our benchmark suite CoBRA available at cobra.cps.cit.tum.de. It is a place to share the generated solutions to the proposed tasks and distribute novel tasks highlighting the properties of different optimizers. The documentation of CoBRA includes detailed descriptions of the abstract objects described within this paper.

Specifically, CoBRA focuses on industrial settings with well-known environments based on synthesized obstacles or 3D scans of actual factory environments. In particular, we include the inherent flexibility in many tasks, such as rotational symmetries of tools, tolerances in execution, or flexibility in the position of the robot's base, into the motion planning problem. Executable robot models are available using Timor [28], including kinematics, dynamics, and collision checking.

Acknowledgements

This work was supported by the ZIM project on energy- and wear-efficient trajectory generation (grant no. ZF4086011PO8) and the EU's Horizon 2020 project CONCERT (grant no. 101016007). We also thank the GrabCAD community and [27] for their published assets and robot models.

[11]cobra.cps.cit.tum.de/ICRA24/example

[12]ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTConnect.html, accessed on Feb. 29th, 2024

REFERENCES

[1] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[2] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Comput. Vision - ECCV*, 2014, pp. 740–755.

[3] C. Goldfeder, M. Ciocarlie, Hao Dang, and P. K. Allen, "The Columbia grasp database," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2009, pp. 1710–1716.

[4] S. Ulbrich *et al.*, "The OpenGRASP benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation," in *IEEE Int. Conf. Intell. Robots Syst.*, 2011, pp. 1761–1767.

[5] J. Mahler *et al.*, "Learning ambidextrous robot grasping policies," *Sci. Robot.*, vol. 4, no. 26, 2019.

[6] C. Chamzas *et al.*, "MotionBenchMaker: A tool to generate and benchmark motion planning datasets," *IEEE Robot. and Automat. Lett.*, vol. 7, no. 2, pp. 882–889, 2022.

[7] B. Cohen, I. Şucan, and S. Chitta, "A generic infrastructure for benchmarking motion planners," in *IEEE Int. Conf. Intell. Robots Syst.*, 2012, pp. 589–595.

[8] M. Moll, I. Şucan, and L. Kavraki, "Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization," *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, pp. 96–102, 2015.

[9] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symp.*, 2017, pp. 719–726.

[10] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede, "A survey on domain-specific modeling and languages in robotics," *J. Softw. Eng. Robot.*, vol. 7, no. 1, pp. 75–99, 2016.

[11] M. Bordignon, K. Stoy, and U. P. Schultz, "Generalized programming of modular robots through kinematic configurations," in *IEEE Int. Conf. Intell. Robots Syst.*, 2011, pp. 3659–3666.

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

[13] S. Chitta, I. Şucan, and S. Cousins, "MoveIt!" *IEEE Robot. Autom. Mag.*, vol. 19, no. 1, pp. 18–19, 2012.

[14] I. Şucan, M. Moll, and L. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, 2012.

[15] Stanford Artificial Intelligence Laboratory et al., *Robotic Operating System*, 2018. [Online]. Available: https://www.ros.org

[16] M. Gorner, R. Haschke, H. Ritter, and J. Zhang, "MoveIt! Task constructor for task-level motion planning," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2019, pp. 190–196.

[17] IFR International Federation of Robotics, *Presentation of World Robotics 2023*. VDMA Robotics + Automation, 2023.

[18] M. Wilson, *Implementation of Robot Systems: An introduction to robotics, automation, and successful systems integration in manufacturing*. Butterworth-Heinemann, 2015.

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

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

[21] E. Icer, H. A. Hassan, K. El-Ayat, and M. Althoff, "Evolutionary cost-optimal composition synthesis of modular robots considering a given task," in *IEEE Int. Conf. Intell. Robots Syst.*, 2017, pp. 3562–3568.

[22] 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 Trans. Robot.*, vol. 34, no. 5, pp. 1240–1251, 2018.

[23] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, "Modular robot design synthesis with deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 10 418–10 425.

[24] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2009, pp. 625–632.

[25] K. M. Lynch and F. C. Park, *Modern Robotics - Mechanics, Planning and Control*. Cambridge University Press, 2017.

[26] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer, 2009.

[27] C. Gaz, M. Cognetti, A. Oliva, P. R. Giordano, and A. De Luca, "Dynamic identification of the Franka Emika Panda robot with retrieval of feasible parameters using penalty-based optimization," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 4147–4154, 2019.

[28] J. Külz, M. Mayer, and M. Althoff, "Timor Python: A toolbox for industrial modular robotics," in *Proc. of the IEEE/RSJ Int. Conf. on Intel. Rob. Sys.*, 2023.