CoBRA: A Composable Benchmark for Robotics Applications

Matthias Mayer, Jonathan Külz, and Matthias Althoff

Abstract— **Selecting an optimal robot and configuring it for a given task is currently mostly done by human expertise or trial and error. To evaluate the automatic selection and adaptation of robots to specific tasks, we introduce a benchmark suite encompassing a common format for robots, environments, and task descriptions. Our benchmark suite is especially useful for modular robots, where the configuration of the robots themselves creates a host of additional parameters to optimize. The benchmark defines this optimization problem and facilitates the comparison of solution algorithms. All benchmarks are accessible through cobra.cps.cit.tum.de, a website to conveniently share, reference, and compare 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]. Another area 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 as well as 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 Parsol[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 robot assemblies for a given task. We provide the first benchmark suite to compare robots and robot assemblies in different real-world environments for various cost functions.

In our literature survey, we give an overview of robot descriptions, the most common robotic tasks in industry, background on modular robot configuration optimization, and state the objectives of our benchmark suite.

TABLE I
Common robot tasks in industry with their market share and predominant actions; market shares from [17]. Point to point movements (PTP) are dominating.

Task	Market Share	Predominant Actions
Handling / Tending	41.0 %	PTP, trajectory
Welding	29.0 %	PTP, trajectory
Assembly	12.0 %	PTP, trajectory, force control
Dispensing	4.0%	trajectory
Predominant Processing	1.4%	trajectory, force control
Other & Unspecified	12.6 %	(indeterminable)

A. Related Work

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

MoveIt! [13] is a common software stack for robotic trajectory 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 with interdependent sub-tasks. Due to the deep integration in MoveIt! their task description is not portable.

2) Typical Robotic Tasks: CoBRA intends to capture the variety of real-world applications a robot today might encounter. Based on an analysis of market shares of different applications in robotics from 2012 [17], these are still mainly in manufacturing; we list the most common robotic tasks in 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

All authors are with the Technische Universität München, Fakultät für Informatik, Lehrstuhl für Robotik und Echtzeitsysteme, Boltzmannstraße 3, 85748, Garching, Germany. {matthias.mayer, jonathan.kuelz, althoff}@tum.de

[1]http://plannerarena.org/, accessed on September 15th, 2022

[2]https://web.archive.org/web/20170503025420/https://parasol.tamu.edu/groups/amatogroup/benchmarks/, accessed on September 15th, 2022

reinforcement learning [23], [24]. The previously-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]. Comparing these optimizers on a common set of tasks and robots enables fair and comparable analysis, particularly for modular robot configuration optimization.

B. Contributions

We introduce CoBRA, a benchmark suite for automatic robot selection, synthesis, and programming, based on well-defined environments, models (robot modules and entire robots), and task descriptions. Perception of and reaction to an unexpected environment are intentionally left out to focus on evaluating robot assemblies for the intended industrial tasks with known and controlled environments. The capability to react to varying circumstances, however, can be enforced through constraints and costs, e.g., by checking whether the robot has sufficient manipulability within an area where bin picking is needed.

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 applications and hand-crafted tasks covering a wide variety of robotic tasks identified in the literature.
- **Portable**: All components of our benchmark suite are described in the JSON markup language, which makes it independent of the platform and programming language. 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 processing tasks. Our robot description is valid for 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 it suitable for any robot design and task.

The remaining paper is organized as follows: In Sec. II we provide the formal definitions of robotic tasks, constraints, and the optimization problem to solve in each benchmark. Later, in Sec. III–V, we describe the implementation of the robots, cost functions, and task descriptions, respectively. Lastly, in Sec. VI, we state our current approach for task generation and provide an example.

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

	Projection Name	Coordinates
Translation $t(\mathbf{T})$	Cartesian	(x, y, z)
	Cylindrical	(r_{cyl}, θ, z)
	Spherical	(r_{sph}, θ, ϕ)
Rotation $R(\mathbf{T})$	XYZ fixed	(r, p, y)
	ZYX Euler	(α, β, γ)
	Axis-angle	$(n_x, n_y, n_z, \theta_R)$
	Quaternion	(a, b, c, d)

II. Task Description and Problem Statement

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

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

$$B = \mathcal{R} : C : \Theta. \quad (1)$$

An example is $\mathcal{R}$ = Panda, C = T (cycle time), S = Factory1 resulting in the benchmark ID `Panda:T:Factory1`. As in CommonRoad [9], parts of this description may be replaced with individual (keyword IND) or modified components (prefix •extitM-), which require auxiliary specifications. Collaborative robot tasks, 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, can be described, too:

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

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

A. Poses

Our benchmarks use poses $\mathbf{T} \in SE(3)$[3] to represent rotations $R(\mathbf{T}) \in SO(3)$[3] and translations $\vec{t}(\mathbf{T}) \in \mathbb{R}^3$ between frames, i.e., a point represented as $\vec{p}_a \in \mathbb{R}^3$ in frame a is represented with respect to frame b by:

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

To constrain and judge the distance between poses, we use a notation similar to [25, Sec. IV.A]: We denote the difference between a pose $\mathbf{T}$ and a desired pose $\mathbf{T}_d \in SE(3)$ after a mapping $S\colon SE(3) \mapsto \mathbb{R}^N, N \in \mathbb{N}^+$ as $\Delta_S(\mathbf{T}, \mathbf{T}_d) = S(\mathbf{T}_d^{-1}\mathbf{T})$. 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 $\mathbf{T}$.

[3] $SO(3)$ is the special orthogonal group representing rotations about the origin in 3D space (3×3 orthogonal matrix with determinant +1); $SE(3) = \mathbb{R}^3 \times SO(3)$ represents both translations and rotations.

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

Most real-world tasks accept some tolerance in execution so that each coordinate i can be between a minimum and maximum value $\gamma_i^{min|max} \in \mathbb{R}$, respectively, which we denote by the stacked intervals $\Gamma(\mathbf{T}_d) = [\gamma_i^{min}, \gamma_i^{max}]_{i=1}^N$; we say $\mathbf{T}$ fulfills $\mathbf{T}_d$ if $\Delta_S(\mathbf{T}, \mathbf{T}_d) \in \Gamma(\mathbf{T}_d)$. 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 $\mathbf{T}_d = I_{4\times4}$ (four-by-four identity matrix) in the world frame, the projections $S(\mathbf{T}) = [r(\mathbf{T}), p(\mathbf{T})]^T$ and intervals $\gamma_{1,2}^{min} = -10^{-3}, \gamma_{1,2}^{max} = 10^{-3}$, respectively.

B. Hybrid Motion Planning Problem

Our benchmark suite extends classical trajectory planning for robotics, which only searches for a trajectory $\vec{x}(t)$ over time t fulfilling a set of goals $\mathcal{G}$ while respecting a set of constraint functions $\mathcal{C}$, detailed later on. We extend this planning problem by

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

The trajectory $x(t) = (\vec{q}(t), \dot{\vec{q}}(t), \ddot{\vec{q}}(t))$ contains the complete state of a (rigid) robot with $n \in \mathbb{N}^+$ joints given by a configuration $\vec{q}(t) \in \mathbb{R}^n$ and its derivative $\dot{\vec{q}}(t)$, as well as the commanded changes $\ddot{\vec{q}}(t)$. We omit the dependency on time, e.g. $\vec{q}(t)$, if not necessary. Based on 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 $\mathbf{T}_{eef}(\vec{q}, M) \in SE(3)$ relative to the base $\mathbf{B}$ [12]; for inverse solutions we use [19, Sec. 2.7];
- robot occupancy $\mathcal{O}(\vec{q}, \mathbf{B}, M) \subset \mathcal{P}(\mathbb{R}^3)$, where $\mathcal{P}(\bullet)$ returns the power set of $\bullet$;
- forward dynamics dyn given control forces $\vec{\tau}$ and the external wrench $\vec{f}^{ext}(t)$: $\ddot{\vec{q}} = dyn(\vec{q}, \dot{\vec{q}}, \vec{\tau}, \vec{f}^{ext}, \mathbf{B}, M)$ [12];
- inverse dynamics $\vec{\tau} = dyn^{-1}(\vec{x}, \vec{f}^{ext}, \mathbf{B}, M)$ [19, Ch. 3.5].

The above functions allow us to define costs J_C for a proposed solution with base pose $\mathbf{B}$, module order M, and a trajectory specified for the time interval $[t_0, t_N]$, by adding terminal costs Φ_C and the integral of running costs L_C:

$$J_C(\vec{x}(t), t_0, t_N, \mathbf{B}, M) = \Phi_C(\vec{x}(t_0), t_0, \vec{x}(t_N), t_N, \mathbf{B}, M) + \int_{t_0}^{t_N} L_C(\vec{x}(t'), t', \mathbf{B}, M) dt'. \quad (4)$$

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

$$[M^*, \mathbf{B}^*, \vec{x}^*] = \underset{M, \mathbf{B}, \vec{x}}{\arg\min}\, J_C(\vec{x}(t), t_0, t_N, \mathbf{B}, M) \quad (5)$$

[5]Extensions to parallel kinematics are discussed in `cobra.cps.cit.tum.de/robot_description`

TABLE III
AVAILABLE CONSTRAINT FUNCTIONS.

Constraint	Constraint function
Joint limits	$\vec{q} \geq \vec{q}_{min} \wedge \vec{q} \leq \vec{q}_{max}$
Joint velocity and acceleration limits	$\lvert\dot{\vec{q}}\rvert \leq \dot{\vec{q}}_{max}, \lvert\ddot{\vec{q}}\rvert \leq \ddot{\vec{q}}_{max}$
Joint torque limits	$\lvert dyn^{-1}(\vec{x}, \vec{f}^{ext}, \mathbf{B}, M)\rvert \leq \vec{\tau}_{max}$
End effector velocity	$\lVert v\rVert_2 \in [v_{min}, v_{max}]$, $\lVert\omega\rVert_2 \in [\omega_{min}, \omega_{max}]$
Keep $\mathbf{T}_{eef}$ close to $\mathbf{T}_d$	$\Delta_S(\mathbf{T}_{eef}, \mathbf{T}_d) \in \Gamma(\mathbf{T}_d)$
Fulfill all goals	$\forall G_i \in \mathcal{G}\, \exists t_{f,i} \in [t_0, t_N]$
Fulfill all goals in order	$\forall G_i, G_j \in \mathcal{G}, i < j : t_{f,i} < t_{f,j}$
No self-collision	$\forall l, l' \in \mathcal{L}, l \neq l' : \tilde{\mathcal{O}}(l) \cap \tilde{\mathcal{O}}(l') = \emptyset$
No collision	$\mathcal{O}(\vec{q}, \mathbf{B}, M) \cap \mathcal{W}_{occ} = \emptyset$
Valid base pose	$\mathbf{B}_{given}$: $\Delta_S(\mathbf{B}, \mathbf{B}_{given}) \in \Gamma(\mathbf{B}_{given})$
Valid assembly	see Sec. III-A

subject to $\forall t \in [t_0, t_N]$:

$$\ddot{\vec{q}} = dyn(\vec{q}, \dot{\vec{q}}, \vec{\tau}, \vec{f}^{ext}, \mathbf{B}, M) \quad (6)$$

$$0 \geq c(\vec{x}, t, \mathbf{B}, M)\ \forall c \in \mathcal{C}. \quad (7)$$

At t_0 the robot must satisfy all constraints in $\mathcal{C}$ (see Sec. II-C) and be stationary, i.e. $\vec{x}(t_0) = (\vec{q}(t_0), \vec{0}_n, \vec{0}_n)$, where $\vec{0}_n$ 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 poses, constraints in $\mathcal{C}$, and goals in $\mathcal{G}$.

C. Constraints

We consider constraints that can be written as $0 \geq c(\vec{x}(t), t, \mathbf{B}, M)$, noting that equality can be ensured by adding the partial constraints $c_i \leq 0 \wedge -c_i \leq 0$. To use Boolean functions b as constraints, 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 $J(\vec{q}) = \frac{dS_d(\mathbf{T}_{eef}(\vec{q}))}{d\vec{q}}$;
- end-effector's velocities $(v, \omega) = J(\vec{q})\dot{\vec{q}}$;
- occupancy of Link l from set of robot links $\mathcal{L}$ as $\tilde{\mathcal{O}}(l)$;
- occupancy of obstacles $\mathcal{W}_{occ}(t)$;
- desired base pose with tolerance $\mathbf{B}_{given}$.

For vectors v, we define the componentwise absolute value as $|v|$ and the 2-norm as $\|v\|_2$. The time to finish a single goal G_i is $t_{f,i}$. With these we show 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 goals $\mathcal{G} = \{G_1, ..., G_N\}$. Each goal $G_i \in \mathcal{G}$ contains a set of constraint functions $\mathcal{C}_i$, a termination condition $g_i(\vec{x})$, and an execution duration $t_{S,i}$. We note that the goal indices i are arbitrary, therefore we can order them such that G_i is fulfilled after G_{i-1}. A goal G_i is fulfilled at time $t_{f,i}$, if $g_i(\vec{x})$ evaluates to

TABLE IV
AVAILABLE GOAL PREDICATES.

Goal	Goal predicate
$at(\mathbf{T}_d, \vec{q})$	$\Delta_S(\mathbf{B}\,\mathbf{T}_{eef}(\vec{q}, M), \mathbf{T}_d) \in \Gamma(\mathbf{T}_d)$
$reach(\mathbf{T}_d, \vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})$	$at(\mathbf{T}_d, \vec{q}) \wedge \|\dot{\vec{q}}\|_2 \leq \epsilon \wedge \|\ddot{\vec{q}}\|_2 \leq \epsilon$
$returnTo(G_i, t)$	$\|\vec{q}(t) - \vec{q}(t_{f,i})\|_2 \leq \epsilon \wedge \|\dot{\vec{q}}(t) - \dot{\vec{q}}(t_{f,i})\|_2 \leq \epsilon \wedge \|\ddot{\vec{q}}(t) - \ddot{\vec{q}}(t_{f,i})\|_2 \leq \epsilon$
$pause(\vec{q}, t_S, t)$	$\forall \tau \in [t - t_S, t] \colon \|\vec{q}(\tau) - \vec{q}(t)\|_2 \leq \epsilon$
$followed(\mathbf{T}(t), \vec{q}, t)$	$t - t_{f,i-1} > t_{S,\mathbf{T}(t)} \wedge \forall \tau \in [0, t_{S,\mathbf{T}(t)}] \colon at(\mathbf{T}(t_{S,\mathbf{T}(t)} - \tau), \vec{q}(t - \tau))$
$left(\mathcal{W}_A, \vec{q}, t_S, t)$	$\forall \tau \in [t - t_S, t] \colon \mathcal{O}(\vec{q}(\tau), \mathbf{B}, M) \cap \mathcal{W}_A = \emptyset$

true and its constraints have been satisfied within its duration $t_{S,i}$: $\forall t \in [t_{f,i} - t_{S,i}, t_{f,i}], \forall c \in \mathcal{C}_i \colon 0 \geq c(\vec{x}(t), t, \mathbf{B}, M)$.

We introduce a small deviation $\epsilon = 10^{-3}$ and a Cartesian trajectory $\mathbf{T}(t)$ describing desired poses for each time step within $[0, t_{S,\mathbf{T}(t)}]$. Other goals may have other explicit duration t_S, such as a pause for t_S seconds, or they take a single time step, e.g., $t_{S,at} = 0$. 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. V). With these prerequisites, we expect that terminal conditions g for most tasks given in Tab. I can be composed from the predicates in Tab. IV.

With respect to Tab. I, *at* and *reach* model simple PTP tasks. *followed* allows one to define more complex trajectory tasks, including external forces. *ReturnTo*, *pause*, and *left* specify synchronization with external machines, such as a CNC mill processing a placed part.

III. Robots

The next three sub-sections will provide implementation details for modeling robots, list the available robots and cost functions, and describe how tasks and proposed solutions are stored.

A. Robot Modeling

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

Each module m has a unique $\texttt{ID}(m)$ within a module set $\mathcal{R}$ with a unique $\texttt{name}(\mathcal{R})$. Any robot with module order M can, therefore, be describes as $\texttt{name}(\mathcal{R})\texttt{.[ID(}m_{1..N}\texttt{)]}$, e.g., see Sec. VI. We integrate Timor Python [26], that provides a transformer from our proposed description to URDF.

Similar to [11], [12], a module is composed of *bodies* and *joints*. Fig. 1 illustrates a single module and Fig. 2 an example of real modules assembled into a robot. Each body specifies the dynamics (such as center of mass *CoM*) and the geometry v (e.g., ellipses in Fig. 1) of a rigid body.

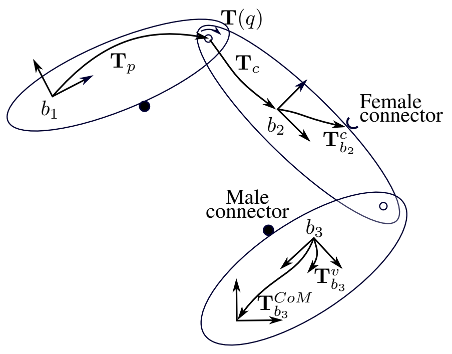

Fig. 1. Sketch of a module consisting of three ellipsoidal bodies b_1, b_2, b_3, each with a body-fixed frame. The bodies are connected via two joints (empty circles) and the joint transformation $\mathbf{T}_p\mathbf{T}(q)\mathbf{T}_c$ is shown between b_1 and b_2. Each body has a connector; an exemplary pose $\mathbf{T}^c_{b_2}$ is shown for body b_2. Body b_3 displays the transformation to its center of mass $\mathbf{T}^{CoM}_{b_3}$ and to its geometry v.

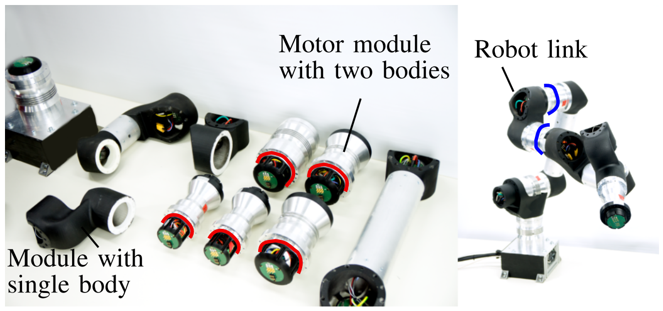

Fig. 2. Modules (left) and links in the assembled robot (right) from the module set PROMODULAR. Modified image from [20, Fig. 1] with red lines marking the separation of bodies in a module (left) and blue lines separating a link in the assembled robot (right).

Additionally, each module provides details about how to connect it to others via *connectors*. Each connector contains lists of supported *size* $\in \mathcal{P}(\mathbb{R})$ and *type* $\in \mathcal{P}(\texttt{string})$. Note that bodies from multiple modules are connected rigidly and together form a link, as known, e.g., from URDF; 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 atleast a common entry in their respective *size* and *type* lists, and their *genders* are both hermaphroditic (genderless) or opposing (male/female); this extends the matching based on gender alone as introduced in [11]. Additionally, each connector c defines its *pose* relative to the body frame b as $\mathbf{T}^c_b$; shown for b_2 in Fig. 1. If two connectors with pose $\mathbf{T}_A, \mathbf{T}_B$, 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[7, Sec. 2.5]. As in [11], allowing arbitrarily many connectors on a module enables us to also model robots with several branches or closed kinematic chains[7].

A joint connects two bodies, referred to as *parent* p and *child* c. Each joint needs three transformations: $\mathbf{T}_p$ from the parent to the joint frame, the joint transformation $\mathbf{T}(q, type)$

[6]`http://wiki.ros.org/urdf/XML`, accessed on September 15th, 2022

[7]`cobra.cps.cit.tum.de/robot_description`,

and $\mathbf{T}_c$, from the joint to the child. The most common type of joint is *revolute* which is a single rotation about the joint frame's z-axis by the angle q; others are given in[7]. An example is visualized between body b_1 and b_2 in Fig. 1. 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 joint torque $\tau_j = I_m k^2 \ddot{q}_j + f_v \dot{q}_j + f_c \, sign(\dot{q}_j)$ [27, Ch. 7].

B. Available Robots

We include models of standard industrial robots that can be used as they are to turn the hybrid motion planning problem into a classical one, because they cannot be reconfigured. Initially, we provide a Stäubli TX-90 [28] and a Kuka LWR 4p [29], [30]. Unifying their formats was done via model fitting as described in [30]. Additionally, we provide a set of robot modules based on Schunk's LWA 4P[8] introduced as IMPROV in [12] and one based on the modular robot proModular.1 used in [20].

IV. 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 solution trajectory to a scalar value or compound functions that weight multiple atomic functions. A detailed description of all available costs can be found on our website[9].

In our example in Sec. VI, we use atomic costs describing the robot mass (J_m), number of actuated joints (J_{numJ}), the time it takes to execute the solving trajectory (J_T) and the mechanical energy required to follow the trajectory (J_{mechE}). In addition, we evaluated the compound cost functions `Liu2` and `Whit2`. `Liu2` weighs mechanical energy and execution time: $J_{Liu2} = J_{mechE} + 0.2 J_T$ [20, Sec. IV B]; `Whit2` the number of actuated joints and the robot mass: $J_{Whit2} = 0.025 J_{numJ} + 0.1 J_m$ [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 cost $J = \sum_{i=1}^{N} w_i J_i$.

V. Tasks

The structure of a task is shown in Fig. 3, describing the scene with its `obstacles`, `constraints` (Sec. II-C) and `goals` (Sec. II-D). Each task is uniquely defined by a `task ID` and a benchmark `version`. Additionally, it contains contact information from the `authors`, `tags` for semantic searches, the `timeStepSize` used for time discretization, and a `date` of publishing. The complete description can be found on our website[10].

There can be stationary obstacles, such as machines or columns, or moving ones with fixed and deterministic trajectories, such as automatic doors. Obstacles are placed at a `pose` relative to the world frame and are represented

```
[1] task
 ├─[1] header
 │  ├─[1] taskID: ℕ⁺
 │  ├─[1] version: string
 │  ├─[1] taskName: string
 │  ├─[1..] author: string
 │  ├─[1..] affiliation: string
 │  ├─[1..] email: string
 │  ├─[0..] tags: string = []
 │  ├─[1] date: string yyyy-MM-dd
 │  ├─[0..1] timeStepSize: float = 0.01
 │  └─[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. 3. (Abbreviated) Structure of the task object. For each attribute we give ranges for the number of allowed elements; fields with a lower bound of `0` elements are optional and come with default values. Fields which 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].

by `collision` and optional `visual` geometries. Those two geometries enable efficient collision checking on simpler over-approximations, while keeping fidelity for visualization if needed. Moving obstacles have time-dependent poses, i.e., a fixed list of poses for each time step of the task.

For every constraint and goal, the task must reference the specific functions in Sec. II-C, II-D via a `type` and give 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_S for pause or $v_{min|max}$ 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 structure of a solution for a robot with a single kinematic chain is provided in Fig. 4. It specifies the task Θ by its `ID` and benchmark `version`, the used `cost function` C, a robot `module set` $\mathcal{R}$ and `module order` M^*, `base pose` $\mathbf{B}^*$, and solving `trajectory` $\vec{x}^*$ as defined in (5). The trajectory is given as a sequence of N samples of $\vec{x} = (\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}})$ at times in `t`. `t_f` states for each goal in a task $G_i \in \mathcal{G}$ its final time $t_{f,i}$.

A solution can be submitted to CoBRA's website[11] where we check whether it solves the task and adheres to the stated constraints. Valid solutions will be published together with provided author information and can be queried for used cost function, final cost, and others.

[8]`https://schunk.com/fileadmin/pim/docs/IM0019885.PDF`, accessed on September 12th, 2022

[9]`cobra.cps.cit.tum.de/solution_description`

[10]`cobra.cps.cit.tum.de/task_description`

[11]`cobra.cps.cit.tum.de/submit`

```
[1] Solution
 ├── [1] taskID: ℕ⁺
 ├── [1] version: string
 ├── [1] costFunction: string
 ├── [1] moduleSet: string
 ├── [1..] moduleOrder: moduleID
 ├── [1] basePose: pose
 ├── [1] cost: ℝ
 └── [1] trajectory
      ├── [1] t: ℝ^N
      ├── [1] q: ℝ^(N×DoF)
      ├── [1] dq: ℝ^(N×DoF)
      ├── [1] ddq: ℝ^(N×DoF)
      └── [1] t_f: Dict(goal_ID → time)
```

Fig. 4. (Abbreviated) structure of the solution object for a robot with a single kinematic chain following the notation from Fig. 3. The complete version is published on our website[9].

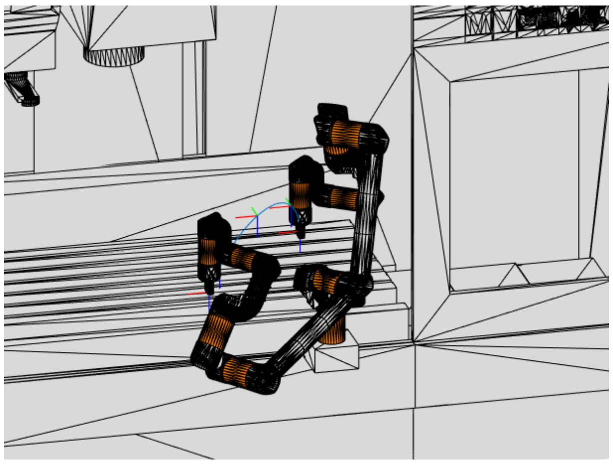

Fig. 5. Best solution trajectory for task PTP_3 (ID: 4) with cost $[(J_T|1)]$. The solving robot was constructed from the set `PROMODULAR` and has the module order $M = [59, 3, 55, 3, 40, 4, 38, 5, 24, 6, 54, 6, 61]$.

VI. Numerical Example

An example task Θ (ID: 4) of our benchmark suite and its solution are shown in Fig. 5. Within Θ, the robot needs to move in and out of a CNC machine. We show the configuration of the robot at the goal poses inside and outside the CNC machine, as well as its end effector trajectory with selected poses to highlight its orientation (Sec. II-D). We include *joint limits* (position and torque), *no collisions*, and a *goal order* as constraints (Sec. II-C).

This task was solved with the robots `PROMODULAR.[59 3 55 3 40 4 38 5 24 6 54 6 61]` (shown in Fig. 5), `KukaLWR4p`, and `IMPROV.[21 31 4 22 32 5 23 33 12]`. Tab. V summarizes different costs C of the generated solution trajectories for each robot. Solutions were generated with OMPL's Rapidly-exploring Random Tree-Connect implementation[12] and adhere to the constraints given in Θ; for each cost, we state the minimal costs found from five generated trajectories.

[12] `https://ompl.kavrakilab.org/classompl_1_1geometric_1_1RRTConnect.html`, accessed on Sept. 15th, 2022

TABLE V
Comparison of minimal solution costs for PTP_3 (ID: 4) with different robots (lower is better).

Cost (from IV)	proModular.1	KukaLWR4p	IMPROV
J_T	4.77	6.92	5.47
J_{mechE}	219.5	162.5	100.4
J_{Whit2}	1.91	2.00	1.62
J_{Liu2}	220.5	163.9	101.5

VII. Conclusions

We introduced CoBRA, the first benchmark suite for finding optimal robotic solutions with conventional and modular robots, which is available at `cobra.cps.cit.tum.de`. CoBRA serves as a place to fairly compare different motion planners and modular robot optimizers. The accompanying documentation includes detailed descriptions for the abstract objects described within this paper, gives access to already submitted and accepts submissions of new tasks, and provides a place to publish and compare the solutions to these tasks.

The benchmark suite focuses industrial settings with well-known environments, which significantly simplifies task description and allows it to disregard perception. In particular, we extended the motion planning problem with the inherent degrees of freedom to many tasks, such as rotational symmetries of tools, tolerances in execution, or flexibility in the position of the robot's base. An executable robot model is available using Timor [26], including kinematics, dynamics and collision checking. In the future, the suite will be extended with more real-world tasks based on 3D scans or design data from real industrial settings.

Acknowledgements

This work was supported by the ZiM project on energy- and wear-efficient trajectory generation (No. ZF4086011PO8) and the EU's horizon 2020 project CONCERT (No. 101016007).

We would like to thank the GrabCAD community and Gaz et al. for their published assets and robot models.

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*, 2014, pp. 740–755.
[3] C. Goldfeder, M. Ciocarlie, Hao Dang, and P. K. Allen, "The Columbia Grasp Database," in *IEEE Int. Conf. Robot. Automat.*, 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, p. eaau4984, 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, p. 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 *IEEE Intell. 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, p. aaw1924, 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] M. Quigley *et al.*, "ROS: an Open-Source Robot Operating System," in *ICRA Workshop on Open Source Softw.*, 2009.

[16] M. Gorner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! Task Constructor for Task-Level Motion Planning," in *IEEE Int. Conf. Robot. Automat.*, 2019, pp. 190–196.

[17] Statistical Department of International Federation of Robotics (IFR), *World Robotics 2012: Statistics, Market Analysis, Forecasts and Case Studies. Industrial robots*, M. Hägele, Ed. VDMA, 2012.

[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 *IEEE Int. Conf. Robot. Automat.*, 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] J. Whitman and H. Choset, "Task-Specific Manipulator Design and Trajectory Synthesis," *IEEE Robot. Automat. Lett.*, vol. 4, no. 2, pp. 301–308, 2019.

[25] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation Planning on Constraint Manifolds," in *IEEE Int. Conf. Robot. Automat.*, 2009, pp. 625–632.

[26] J. Külz, M. Mayer, and M. Althoff, "Toolbox for Industrial Modular Robotics: Timor Python," arXiv:2209.06758 [cs.RO], 2022.

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

[28] J. Jin and N. Gans, "Parameter Identification for Industrial Robots with a Fast and Robust Trajectory Design Approach," *Robot. and Comput.-Integr. Manuf.*, vol. 31, no. 1, pp. 21–29, 2015.

[29] A. Jubien, M. Gautier, and A. Janot, "Dynamic Identification of the Kuka LWR Robot Using Motor Torques and Joint Torque Sensors Data," in *IFAC Proceedings Volumes*, 2014, pp. 8391–8396.

[30] C. Gaz, F. Flacco, and A. De Luca, "Extracting Feasible Robot Parameters from Dynamic Coefficients Using Nonlinear Optimization Methods," in *IEEE Int. Conf. Robot. Automat.*, 2016, pp. 2075–2081.