cRoK: A Composable Robotics Benchmark

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 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 creation of the robots themselves creates a host of additional parameters to optimize. The benchmark defines this optimization and facilitates the comparison of solution algorithms. All benchmarks are accessible through 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 vision to tasks of varying difficulty from single, cropped frame labeling to detecting multiple objects. These data sets certainly coincided with the resurgence of (deep) learning and possibly enabled it in the first place [2].

Applications of deep learning to the robotic tasks of grasping and/or bin picking have been published in [3]–[5]; more have been 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 general planning community, only a few benchmarks are established, e.g., by the Open Motion Planning Library (OMPL) [7], [8] 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, common robotic tasks in industry, background on modular robot optimization, and state the goals of our benchmark.

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 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! the task description of [16] is not portable.

2) Typical Robotic Tasks: cRoK 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 industry; 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., visually, for grasp planning, quality assessment, and reworking.

3) Optimizing Modular Robots: One key aspect we want to address with our benchmark suite is modular or reconfigurable robotics [19, Ch. 22.2], where even small module sets can be used to assemble millions of possible robots requiring efficient task planning [20]. Recent solutions have been developed using hierarchical elimination combined with

TABLE I

| Task                  | Market Share [17] | Predominant Actions                        |
|----------------------|-------------------|---------------------------------------------|
| Handling / Tending   | 41%               | PTP, trajectory                             |
| Welding              | 29%               | PTP, trajectory                             |
| Assembly             | 12%               | PTP, traj., force control                   |
| Dispensing           | 4%                | Trajectory                                  |
| Predominant Processing| 1.4%              | Trajectory, force control                   |
| Other & Unspecified  | 12.6%             | (indeterminable)                            |
kinematic restrictions [12], [20], genetic algorithms [21], heuristical search [22], and machine learning [23], [24]. The previously-mentioned methods use modules and create modular robots with significantly different properties whose complete specification is often not published. Therefore, comparisons of optimization methods are either not done at all, e.g. [12], [22], or are done on self-implemented versions of other’s work on their own set of modules, e.g. [23], making the comparison difficult. Comparing these optimizers on a common set of scenarios and robots enables improved analysis of optimization strategies in general and modular robot synthesis in particular.

B. Contributions

We introduce a benchmark suite for automatic robot selection, synthesis, and programming, based on well-defined environments, models of robot modules or entire robots, and task descriptions. Perception and reaction to a changing environment are intentionally left out so that we obtain deterministic solutions. The capability to react to varying circumstances, however, can be enforced through constraints and costs, e.g., by checking that the robot has sufficient manipulability within an area where bin picking is needed.

Our benchmark 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, scenarios, and cost functions) allows one to easily compose new benchmarks by recombining existing components.
- **Representative**: The benchmark suite contains real applications and hand-crafted scenarios covering a wide variety of robotic tasks identified in the literature.
- **Portable**: All components of the 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**: Scenarios can describe simple pick and place tasks as well as complex processing tasks. The 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**: The 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 robot manufacturer’s product suites, making it suitable for comparisons and interchange of robot designs and scenarios.

The remaining paper is organized as follows: In Sec. [I], we provide the formal definitions of robotic tasks, constraints, and the optimization problem to solve in each benchmark. Later, in Sec. [III] we describe the implementation of the robots, cost functions, and scenario descriptions, respectively.

Lastly, in Sec. [VI] we state our current approach for scenario generation and give an example.

II. Task Description and Problem Statement

Each benchmark is composed of a set of robot modules $R$, a cost function $C$, and a scenario $S$. Each scenario itself includes an environment $E$ and multiple tasks $T$. The module set $R$ may only contain a single robot or a set of modules with multiple valid combinations.

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

$$B = R : C : S.$$  

(1)

An example is $R = \text{Panda}$, $C = \text{TCyc}$, $S = \text{Fac1}$ resulting in the benchmark ID $B = \text{Panda:TCyc:Fac1}$. As in CommonRoad [9], parts of this description may be replaced with individual (keyword IND) or modified components (prefix M-), which need further explanation. Collaborative robot tasks, where $n \in \mathbb{N}^+$ robots work in a common scenario $S$ and each robot fulfills tasks with its individual cost function $C_1$ to $C_n$, can be described, too:

$$B = [R_1, ..., R_n] : [C_1, ..., C_n] : C-S.$$  

(2)

Note that the prefix $C$ of scenario $S$ indicates that it is collaborative; the prefix $M$ precedes $C$.

A. Hybrid Motion Planning Problem

Our benchmark suite extends classical trajectory planning for robotics, which only searches for a trajectory $x(t)$ fulfilling a task $T$ while respecting a set of constraint functions $G$, detailed later on. We extend this planning problem by a) synthesizing robots from modules, resulting in an ordered set of modules $M = (m_1, ..., m_N)$, $m_n \in R$ to be assembled for serial kinematics, and b) additionally optimizing the base pose $B \in SE(3)$. Note that if the available module set $R$ only includes one valid robot and $B$ is restricted to a single pose, the hybrid motion planning problem is reduced to standard motion planning for robotic manipulators.

Given the set of previously completed goals $(T_1, ..., T_I), T_i \in T$, which may have altered the robot, e.g., with a picked object, we can construct a robot model, including kinematics and dynamics [12]. This model includes functions to map the state $x(t) = (q(t), \dot{q}(t), \ddot{q}(t))$ to

- an end effector pose $P_{eff} = f_k(q, M)$ relative to the base $B$; for inverse solutions we use [19, Sec. 2.7];
- a robot occupancy $O_f(q, B, M) \subset \mathcal{P}(\mathbb{R}^3)$, where $\mathcal{P}(\bullet)$ returns the power set of $\bullet$;
- forward dynamics given control forces $\tau$ and external wrench $F_{ext}: \ddot{q}(t) = dyn_1(q, \dot{q}, \tau, F_{ext}, B, M)$;
- inverse dynamics $\tau = dyn_1^{-1}(x, F_{ext}, B, M)$ [19, Sec. 3.5].

Extensions to parallel kinematics are discussed in [crok.cps.in.]

tum.de/robot_description
These functions are convenient for defining costs $J_C$ for the time frame $[t_0, t_f]$, base pose $B$, and module order $M$, as the sum of terminal costs $\Phi_C$ and running costs $L_C$:

$$J_C(x(t), t_0, t_f, B, M) = \int_{t_0}^{t_f} \Phi_C(x(t'), t, t, B, M)dt'.$$

Formally, the hybrid motion planning problem is to find a module order $M^*$, base placement $B^*$, and trajectory $x^*$ minimizing a given cost $J_C$:

$$[M^*, B^*, x^*] = \arg \min_{M, B, x} J_C(x(t), t_0, t_f, B, M)$$

subject to $\forall t$:

$$q(t) = \text{dyn}_t(q(t), \dot{q}(t), \tau(t), F_{ext}(t), B, M)$$

$$0 \geq g(x(t), t, B, M) \forall g \in \mathcal{G}$$

The initial robot state can be any stationary one, i.e. $x(t_0) = (q(t_0), 0, 0)$, satisfying all constraints in $\mathcal{G}$. In the next sections we will introduce our definitions for poses, constraints in $\mathcal{G}$, and tasks $T$.

B. Poses

Our definitions need poses $P \in SE(3)$, that can represent rotations $R(P) \in SO(3)$ and translations $t(P) \in \mathbb{R}^3$ between frames, i.e., a point $p_a \in \mathbb{R}^3$ in frame $a$ is represented with respect to frame $b$ by $p_b$:

$$p_b = P_a^b p_a = R(P_b^a)p_a + t(P_b^a)$$

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 $P$ and a desired pose $P_d$ after a mapping $S: SE(3) \rightarrow \mathbb{R}^n, n \in \mathbb{N}^+$ as $\Delta_S(P, P_d) = S(P_d^{-1}P)$. $S$ is the Cartesian product of any of the projections listed in Tab. II, e.g., the default $S(P) = [x(P), y(P), z(P), \theta_R(P)]$ contains the three Cartesian coordinates and the rotation angle about an axis for any pose $P$.

Most real-world tasks accept some tolerance in execution, i.e., the executed pose $P_d$ can deviate from the desired pose $P_d$ in each coordinate between a minimum $c_{min} \in \mathbb{R}$ and maximum value $c_{max} \in \mathbb{R}$, which we denote by the interval $C(P_d) = [c_{min}, c_{max}]^n$, i.e., we consider $P$ and $P_d$ the same if $\Delta_S(P, P_d) \in C(P_d)$. If not defined otherwise, we default to $C(P_d) = [-\epsilon, \epsilon]^n$, where $\epsilon = 10^{-3}$. As an example, one may want to constrain the end effector to be upright (z pointing upwards), which can be done by using $P_d = I_{4 \times 4}$ in the world frame, $S(P) = [r(P), p(P)]^T$, and $C(P_d) = [-\epsilon, \epsilon]^2$.

C. Task Constraints

In contrast to the goal’s termination condition introduced later, constraints have to hold at every time step. We consider constraints that can be written as $g(x(t), t, B, M) \leq 0$ and equality can be ensured by adding the partial constraints $g_i \leq 0 \land \neg g_i \leq 0$. To use Boolean functions $b$ as constraints, we use the operator $\land(b)$, which evaluates to $-1$ if $b$ is true and otherwise to 1. We allow to split $\mathcal{G}$ into constraints holding for an entire task $T$, named $\mathcal{G}_T$, and those applying to an intermediate goal $T_i$ named $\mathcal{G}_i$.

Let us introduce the componentwise absolute value of a vector $v$ as $|v|$, the 2-norm as $||v||_2$, and the time to finalize a task $T$ or goal $T_i$ as $t_f(T)$ and $t_f(T_i)$. We can now define the following constraints to formalize $\mathcal{G}_T$ and $\mathcal{G}_i$:

- Joint limits: $q \geq q_{lower} \land q \leq q_{upper}$
- Joint velocity / acceleration limits: $|\dot{q}| \leq \dot{q}_{max}$, $|\ddot{q}| \leq \ddot{q}_{max}$
- Joint torque limits: $|\text{dyn}_t^{-1}(x, F_{ext}(t), B, M)| \leq \tau_{max}$
- End effector velocity: $v_{min} \leq ||\omega||_2 \leq v_{max}$, $\omega_{min} \leq ||\omega||_2 \leq \omega_{max}$
- Valid assembly: see Sec. III-A
- No collision: $\forall l, l \neq l', O(l) \cap O(l') = \emptyset$
- Valid base pose $B$: $\Delta_S(B, B_T) \in C(B_T)$
- Valid assembly: see Sec. III-A

D. Tasks

In every scenario $S$, each robot has to fulfill a task $T$ given as an ordered set of (sub-)goals $T = (T_1, ..., T_N)$. Each goal $T_i$ contains a set of constraint functions $\mathcal{G}_i$ and a termination condition $T_i(x)$. A goal $T_i$ is fulfilled if $T_i(x)$ evaluates to true and its constraints have been satisfied since the termination of the previous goal $T_{i-1}$: $\forall t \in [t_f(T_{i-1}), t_f(T_i)], \forall g \in \mathcal{G}_i$: $g(x(t), t, B, M) \leq 0$. Each goal fulfillment may also change the robot’s occupancy $O_t$ and dynamics $\text{dyn}_t$ due to picking or releasing objects.

We introduce a pose $P_e \in SE(3)$, which the end effector must match to grasp an object $o$, as well as the current time $t$, and a duration $t_g$. A Cartesian trajectory $P(t)$ describes
poses for each time step within $[0, t_f(P(t))]$. Furthermore, the propositions \textit{open}(x, M) and \textit{close}(x, M) return whether the end-effector is open or closed. We define $W_A \subset \mathcal{P}(\mathbb{R}^3)$ as a set of already occupied space, e.g., the occupancy of the interior of a machine (see Sec. III-A). Additionally, a tool held with the end effector works on the space $\mathcal{O}_T(q) \subset \mathcal{P}(\mathbb{R}^3)$, e.g., a drill held in the end-effector could be represented by a cylinder along the end effector’s z-axis, which describes the geometry of the drilled hole.

With these prerequisites, we expect that terminal conditions for most tasks given in Tab. I can be composed from the following predicates:

- $at(P, q) \Leftrightarrow \Delta_S(BP_{\text{ef}}(q, M), P) \in C(P)$
- $\text{reach}(P, x) \Leftrightarrow at(P, q(x)) \land \|q(x)\|_2 < \epsilon \land \|\dot{q}(x)\|_2 < \epsilon$
- $\text{returnTo}(q, t, s) \Leftrightarrow \|q(t) - q(t-s)\|_2 < \epsilon \land \|\dot{q}(t) - \dot{q}(t-s)\|_2 < \epsilon$
- $\text{pause}(q, t, s) \Leftrightarrow \forall \tau \in [t-s, t]: \|\tau - q(\tau)\|_2 < \epsilon$
- $\text{picked}(obj, P_{obj}, x) \Leftrightarrow \text{reach}(P_{obj}, x) \land \text{close}(x, M)$
- $\text{placed}(obj, P_{obj}, x) \Leftrightarrow \text{reach}(P_{obj}, x) \land \text{open}(x, M)$
- $\text{followed}(P, x, t) \Leftrightarrow t - t_f(P(t)) < \epsilon$
- $\text{left}(W_A, q, t, s) \Leftrightarrow \forall \tau \in [t-s, t]: \mathcal{O}(q, \tau, B, M) \cap W_A = \emptyset$
- $\text{covered}(W_A, \mathcal{O}(q, \tau, t)) \Leftrightarrow W_A \subset \bigcup_{\tau \in [t_f(T_{\tau-1}), t]} \mathcal{O}(q(\tau))$

### III. Robots

The next three sub-sections will provide implementation details for the robot model, list the available robots and cost functions, and describe how scenarios are stored.

#### A. Robot Modelling

For robots, we propose a modeling format similar to the universal robot description format (URDF), where each robot consists of modules that can be assembled via connectors. Therefore, we combine the kinematic model from [11] with our previous work in [12]. Each module is structured as shown in Fig. 1 and its implementation is detailed on our website [13]. For convenience, we provide a transformer for serial chains from our description to URDF [14].

Each module is part of a module set $\mathcal{R}$ and has a unique ID within this set, which lets us describe any kinematic chain by referring to the module set and enumerating the module’s IDs from the base to the end effector, e.g., see Fig. 6. We decided on this two-level ID, as this allows module designers to select their IDs arbitrarily and we do not allow combining modules from different sets.

Similar to [11], a module is made out of bodies and joints, which also extend URDF’s links and joints, respectively; see Fig. 2 for a single module and Fig. 3 for an example of real modules assembled into a robot. Each body specifies the dynamics and geometrical properties of a rigid body and provides details about how to connect this module with other modules.

---

[1] crok.cps.in.tum.de/robot_description
[2] crok.cps.in.tum.de/urdf_transformer

---

Fig. 1. Abbreviated structure of the Module object from crok.cps.in.tum.de/robot_description. For some fields we provide default values.

Fig. 2. Sketch of a module consisting of three ellipsoidal bodies $b_1, b_2, b_3$, each with a body frame. The bodies are connected via two joints (empty circles) and the joint transform $P_\beta P(q)P_\alpha$ is shown between $b_1$ and $b_2$. Every body has a connector, with an exemplary pose $P_{\text{b}3}$ shown for body $b_3$. Body $b_3$ displays the center of mass frame $CoM(b_3)$.

Fig. 3. Modules (left) and links in the assembled robot (right) from the module set ModRob-V1. Modified image from [20, Fig. 1] with red lines marking the separation of bodies in a module or links in the assembled robot.
to others via connectors. Each connector is similar to a connection in [11] and defines its pose relative to the body frame $P_c^b$, as seen on $b_2$ in Fig. 2, and specifies a size, gender, and type. A robot assembly is valid if connectors match, i.e., they share a common size and type, and their genders are both hermaphroditic (gender-less) or opposing (male/female). If two connectors with pose $P_{A/B}$, relative to body $A$ and $B$, are connected during assembly, this results in a rigid transform $P_A P_B^{-1}$ mapping points in the body frame of $B$ to $A$’s. As in [11], allowing arbitrarily many connectors on a module enables us to also model robots with several branches or closed kinematic chains, which we discuss on our website.

The dynamic parameters of each body $b$ have the same structure as URDF specifying mass $m(b)$, a frame $\mathbf{r}_{\text{com}}$ whose origin is the center of mass $\text{CoM}(b)$, and an inertia matrix relative to the CoM frame $I(b)$; the inertial frame is shown for $b_3$ in Fig. 2. Modules are rigidly connected during assembly resulting in links $L = \{l_1, \ldots, l_n\}$, which each consist of bodies $B(l_i) = \{b_1, \ldots, b_n\}$. Within each link $l_i \in L$, we can find transformations from the frame of each body $b_j \in B(l_i)$ to a common link frame, i.e., $P_{l_i}^b$. Following [12], this results in:

$$m(l_i) = \sum_{b \in B(l_i)} m(b) \tag{8}$$

$$\text{CoM}(l_i) = \frac{1}{m(l_i)} \sum_{b \in B(l_i)} P_{l_i}^b \text{CoM}(b) \tag{9}$$

$$I(l_i) = \sum_{b \in B(l_i)} R_{l_i}^{b} I(b) \left[R_{l_i}^{b}\right]^{-1} + m(b)\left[p_{l_i}^b | x | p_{l_i}^{b T}\right] \tag{10}$$

The total mass is summed, the CoM is determined by a weighted sum and the total inertia tensor via Steiner’s Theorem. Here $[\bullet]_x$ denotes the mapping of $\bullet$ to the skew-symmetric matrix equaling to the cross-product, i.e., $\forall v \in \mathbb{R}^3, [\bullet]_x v = \bullet \times v$.

The specification of a module’s geometry mirrors URDF allowing us to define sets of visual $\mathcal{V}(b)$ and occupancy geometries $\mathcal{O}(b)$ for each body $b$, shown as ellipses in Fig. 2. The split into visual and occupancy geometries enables us to consistently and efficiently test for collisions while retaining details during visualization. Here, each $v \in \mathcal{V}(b), o \in \mathcal{O}(b)$ is either a simple geometric primitive (box, cylinder, sphere) or a triangulated mesh; each may be transformed by $P_{l_i}^b(v/o)$ relative to the body frame. $v, o$ can be seen as the set of points contained within the geometries. As with the dynamics, the assembled robot has links $l_i \in L$ with overall visual geometries and occupancies:

$$\mathcal{V}(l_i) = \bigcup_{b \in B(l_i)} \bigcup_{v \in \mathcal{V}(b)} P_{l_i}^b P_{l_i}(v)v \tag{11}$$

$$\mathcal{O}(l_i) = \bigcup_{b \in B(l_i)} \bigcup_{o \in \mathcal{O}(b)} P_{l_i}^b P_{l_i}(o,o) \tag{12}$$

Joints mostly work as in URDF, connecting bodies within a module. URDF’s Joints have been extended by a pose relative to the parent $P_p$, as well as, child body $P_c$ frame. With the joint inherent transformation $P_j(q, type)$, this results in the overall transformation $P_j = P_p P_j(q, type) P_c$. So far, we use revolute and prismatic joint types, which allow rotations about or translations along the joint’s $z$-axis, respectively.

Additionally, joints may be passive, which helps to model robots with closed kinematic chains. We also extended the joint’s properties making it possible to model a gear ratio $k$, motor side inertia $I_m$, as well as Coulomb $f_c$ and viscous friction $f_v$, resulting in an additional motor load of $\tau_j = I_m k^2 \dot{\theta} + f_c \dot{\theta} + f_v \text{sign}(\dot{\theta})$ [26, Ch. 7].

### B. Available Robots

We include models of standard industrial robots that can be used as-is to turn the hybrid motion planning problem into a classical one, as they cannot be reconfigured. Initially, we provide a Staubli TX-90 [27], and Kuka LWR 4p [28], [29]. Unifying their formats was done via model fitting as described in [29]. Additionally, we provide a set of robot modules based on Schunk’s Powerball, used in [12], and one based on the modular robot ModRobV1 introduced in [20].

### IV. COST FUNCTION

We have reviewed the literature and found the following partial cost functions used in (modular) robot optimization, which can be used within our benchmark suite:

- **Trajectory time** $T$: $T = t_f - t_0$ [30]
- **Distance**
  - Linear $\text{dist}_{\text{lin}}$ [31, Eq. 4]: $\|q_{\text{desired}} - q_{\text{ref}}(t_f)\|
  - Angular $\text{dist}_{\text{ang}}$ [31, Eq. 5]: $\theta_R(R_{\text{desired}} R_{\text{rec}}^{-1}(t_f))$
  - Joint space $\text{dist}_{\text{q}}$ [21, Eq. 8]: $\|q(t_0) - q(t_f)\|$  
  - Obstacle proximity $\text{obs}_\text{proxim}$ [21, Eq. 9]: $(1 + \delta)^{-1}$, with $\delta$ returning the radius of an enclosing sphere around the object $\bullet$, and $\delta = \text{argmin}_{b \in B, o \in \mathcal{O}(b)} \text{dist}(b, o) - (\text{dist}(o) + r(b))$
- **Reachability** $\text{reach}$ [31, Eq. 3], [21, Eq. 3]:
  - $J_R = \sum_{q \in \text{PoseGoals}} \frac{\text{dist}(q, \text{base}) - \text{dist}(q, \text{arm})}{\text{dist}(q, \text{base})}$
  - $J_{\text{Larm}} = \sum_{l \in \text{link}(M)} \text{length}(l)$
- **Mechanical Energy** $\text{mechE}$ [20, Eq. 2]:
  - $\int_{t_0}^{t_f} \|\dot{q}\|^2 dt$
  - Traveled distance $\text{qdist}$: $\int_{t_0}^{t_f} \|\dot{q}\| dt$; may just be between key poses as in [22]
- **Manipulability** (within subset of $\text{SE}(3)$)
  - Mean dexterity index in workspace over $M$ uniform sampled positions manLiu [20, Eq. 8]
  - Number of reachable poses from a set $\text{reachN}$ [23, Eq. 5]
  - Percentage of reachable poses from set $\text{reachP}$ [21, Eq. 10]

---

9 According to the authors, they used the percentage of solved inverse kinematics for a uniform grid with sample distance 0.1 (Workspace 1) or 0.05 (W.S. 2, 3) and varying orientations at each pose (diagonal in each of the octant directions with 4 roll orientations at ±45, ±135 degrees).
### TABLE III

**OVERVIEW OF COMPOSITE COST FUNCTIONS FOR ROBOT CONFIGURATION AND TRAJECTORY OPTIMIZATION.**

| ID | Partial cost functions and weights | Reference |
|----|-----------------------------------|-----------|
| Liu1 | $(mechE[1])$ | [20, Eq. 2] |
| Liu2 | $(mechE[1]), (T[0.2])$ | [20, Sec. IV B] |
| Liu3 | $(manLiu[1])$ | [20, Eq. 8, 9] |
| Whit1 | $(numJ[0.025], (mass[0.1]), (reach.N[1]))$ | [23, Eq. 5] |
| Whit2 | $(numJ[0.025], (mass[0.1]))$ | [23, Tab. 1] |
| Whit3 | $(reach.N[1], numTrial)$ | [23, Tab. 1] |
| Ha1 | $(volume[1], (numJ[0.3]), (qdist[0.001]))$ | [22, Tab. 2. Manip.] |
| Ha1 | $(volume[1], (numJ[0.1]), (qdist[0.001]))$ | [22, Tab. 2. Legged] |
| Icer1 | $(reach[1/T], (dist.in[1/T]), (dist.ang[1/T]), (dist.det[1/T]), (inv.mod[1/T]), (proxim[1/T]))$ | [21, Eq. 10] |
| Icer2 | $(reachP[0.5], (Icer1[0.5]))$ | [21, Eq. 10] |

- Weight carrying capacity at robot flange $loadC$ [18, Ch. 2.2]
- Dexterity at goal pose $dex$ [31, Eq. 10f]:
  \[ 1 + \frac{1}{det(JJ^T)} \]
- Goal-post test $q_{post}$ (time to move 25 mm up, 300 mm sideways, and 25 mm down; usually for SCARA) [18, Ch. 2.1.2]

Additionally, most optimized a robot’s estimated production cost, by penalizing the total mass $mass$ and production cost, by penalizing the total volume $volume$ [22, Eq. 1, $gs$], number of actuated joints $numJ$ [23, Eq. 5, $N_J$], [22, Eq. 1, $g_J$], or the length of the used modules $invmod$ [21, Eq. 7]:
\[
\sum_{i=1}^{N} length(m_i) + is_{prismatic}(m_i) + q_{upper}(m_i) \]
\[
\frac{dist(goal,B)}{dist(goal,B)}
\]
However, we suggest to explicitly model the cost of each module, e.g., using the purchasing price.

These partial cost functions focusing on single aspects of the robot can be linearly combined to optimize different aspects of the robot. We denote a weighted sum of $N$ cost functions as $[J_1[w_1], ..., J_N[w_N]]$ to define a total cost $J = \sum_{i=1}^{N} w_i J_i$. Total cost functions from recent contributions to modular robot optimization are listed in Tab. III if these were maximized, such as a fitness value, we neglected their weight such that they are costs to be minimized.

### V. SCENARIOS

The structure of a scenario is shown in Fig. 4 describing the scene with its obstacles and the tasks as introduced in Sec. II-D. Each scenario is uniquely defined by a scenarioID and a benchmark version. Additionally, it contains author information, tags for semantic searches, the time step size used for time discretization, and a date of publishing. The complete description can be found on our website [12].

There can be stationary obstacles, such as machines or columns, or moving ones with fixed and deterministic trajectories, such as automatic doors. Extensions to unknown or stochastic behavior, such as a human simulator as described in [32], are conceivable but not yet included.

Obstacles are represented by a geometry (mesh, box, sphere, cylinder) placed at a pose relative to the world frame. Moving objects have time-dependent poses, i.e., a fixed list of poses for each time step of the scenario.

For each robot in the scenario, there is a task that can contain objects to interact with, a valid base pose, constraints to obey and goals to fulfill. Each Task has a scenario-wide unique ID. An object can have the same parameters as a module’s body to state geometric and dynamic properties, as well as connectors which specify where it may be picked up ($P_o$ in Sec. II-D). Pick / place goals specify attaching / detaching of objects on completion, altering the robot. Additionally, one may specify an external wrench on the end effector $F_{ext}$ during execution, e.g., from a tool.

For every constraint and goal in Fig. 4 the scenario must refer to the previously defined functions in Sec. II-C and II-D. These are parameterized by poses $P$, objects $obj$ (via ID), or regions $W_A, R_T$, which can be specified as polylines, i.e., an ordered set of positions connected via straight lines, or any geometry, as defined for a module’s body.

The structure of a solution file is given in Fig. 5. It has to state the scenario $S$ by ID and benchmark version, the used cost function $C$, a robot module set $R$ and order $M^*$, base pose $B^*$, and solving trajectory $x^*$ as defined in (4). The trajectory is given as arrays for the robot’s state $x = (q, \dot{q}, \ddot{q})$ over time $t$.

A solution can be submitted to our website [12] where we check whether it solves the scenario and adheres to the

---

10We interpreted the exponential function as fitness scaling to improve GA performance and therefore left it out.

12[crok.cps.in.tum.de/submit]
stated constraints. Valid solutions will be published together with provided authorInfo (name, E-Mail, affiliation, and publication) and can be searched for by used cost-function, final cost, etc.

VI. BENCHMARK GENERATION AND EXAMPLE

For the first scenarios, we resort to partially manual generation similar to [20]. We searched for machine models online and found multiple on grabcad.com. For each of these, we define areas of interest, such as their workspace or locations of the tool changer. We then applied path planning to some of the robots mentioned in Sec. III-B between different sampled poses in the areas of interest. If a path between poses is found we consider them valid goals for a scenario. These scenarios approximate machine tending tasks, such as pick and place of parts or tools. Secondly, we used available geometries of parts, such as an aircraft window and extracted contours, which would be interesting to follow, such as the window seam. We then validated these contours as feasible paths by planning trajectories along them with one of our robot models.

An example scenario S (ID: 4) and its solution are shown in Fig. 6. Within S, the robot needs to move in and out of a CNC machine. We show the state of the robot at the reached poses in- and outside the CNC machine, as well as its end effector trajectory with selected poses to highlight its orientation.

This scenario was solved with the robots ModRobV1. [59 3 55 3 40 4 38 5 24 6 54 6 61] (shown in Fig. 6), Kuka LWR 4p, and Schunk. [21 31 4 22 32 5 23 33 12]. Tab. [IV] highlights how easy it is to summarize a generated solution trajectory for each robot with respect to different costs C. Solutions were generated with OMPL’s RRT implementation and adhere to the constraints given in S; for each cost we state the minimum one found out of five generated trajectories.

| Cost | ModRobV1 | Kuka | Schunk |
|------|----------|------|--------|
| T    | 4.77     | 6.92 | 5.47   |
| mechE| 219.5    | 162.5| 100.4  |
| qdist| 6.28     | 14.34| 8.71   |
| Whit2| 1.91     | 2.00 | 1.62   |
| Liu2 | 220.5    | 163.9| 101.5  |

VII. CONCLUSIONS

We introduced the first benchmark suite for finding optimal robotic solutions with conventional and modular robots, which is available at crok.cps.in.tum.de. Here, for the first time, we provide a place to fairly compare different robots and motion planners. Our website includes detailed descriptions for the abstract objects described within this paper, gives access to and allows submission of new scenarios, and provides a place to publish and compare the solutions to these scenarios.

We plan to extend the benchmark suite with utilities, such as an executable robot model with kinematics, dynamics, and collision checking, as well as, a baseline optimizer for selecting robots from a module set and generating solving trajectories for each of the tasks. Furthermore, we hope to integrate more real-world tasks based on 3D scans or design data of real industrial settings.

ACKNOWLEDGEMENTS

This work was supported by the ZiM project on energy- and wear-efficient trajectory generation (No. ZF408601P08) and the EU’s horizon 2020 project CONCERT (No. 101016007).

We would like to thank the GrabCAD community, Stäubli International AG and Gaz et al. for giving permission to share their assets and robot models.
REFERENCES

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision (IJCV), vol. 115, no. 3, pp. 211–252, 2015.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: common objects in context,” in Computer Vision – ECCV, 2014, pp. 740–755.

[3] C. Goldfeder, M. Ciocarlie, Hao Dang, and P. K. Allen, “The Columbia grasp database,” in IEEE International Conference on Robotics and Automation, 2009, pp. 1710–1716.

[4] S. Ulbrich, D. Kappler, T. Asfour, N. Vahrenkamp, A. Bierbaum, M. Przybylski, and R. Dillmann, “The OpenGRASP benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation,” in IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011, pp. 1761–1767.

[5] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, “Learning ambidextrous robot grasping policies,” Science Robotics, vol. 4, no. 26, p. eau4984, 2019.

[6] C. Chamzas, C. Quintero-Pelia, Z. Kingston, A. Orthy, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, “MotionBenchMaker: A Tool to Generate and Benchmark Motion Planning Datasets,” IEEE Robotics and Automation Letters - PREPRINT, 2022.

[7] B. Cohen, I. Şucan, and S. Chitta, “A Generic Infrastructure for Benchmarking Motion Planners,” in Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 589–595.

[8] M. Moll, I. Şucan, and L. Kavraki, “Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization,” IEEE Robotics & Automation Magazine, 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 Intelligent Vehicles Symposium, 2017, pp. 719–726.

[10] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede, “A Survey on Domain-specific Modeling and Languages in Robotics,” Journal of Software Engineering in Robotics, 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 International Conference on Intelligent Robots and Systems, 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,” Science Robotics, vol. 4, no. 31, p. aaw1924, 2019.

[13] S. Chitta, I. Şucan, and S. Cousins, “MoveIt!” IEEE Robotics & Automation Magazine, vol. 19, no. 1, pp. 18–19, 2012.

[14] I. Şucan, M. Moll, and L. Kavraki, “The Open Motion Planning Library,” IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72–82, 2012.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, “ROS: an open-source Robot Operating System,” in ICRA Workshop on Open Source Software, 2009.

[16] M. Gorner, R. Haschke, H. Ritter, and J. Zhang, “MoveIt! task constructor for task-level motion planning,” in IEEE International Conference on Robotics and Automation, 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. Hägèle, 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 International Conference 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/RSJ International Conference on Intelligent Robots and Systems, 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 Transactions on Robotics, 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 Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 06, apr 2020, pp. 10418–10425.

[24] J. Whitman and H. Choset, “Task-Specific Manipulator Design and Trajectory Synthesis,” IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 301–308, 2019.

[25] D. Berenson, S. S. Srinivas, D. Ferguson, and J. J. Kuffner, “Manipulation planning on constraint manifolds,” in IEEE International Conference on Robotics and Automation, 2009, pp. 625–632.

[26] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, Robotics, set. Advanced Textbooks in Control and Signal Processing. Springer, 2010.

[27] J. Jin and N. Gans, “Parameter identification for industrial robots with a fast and robust trajectory design approach,” Robotics and Computer-Integrated Manufacturing, vol. 31, no. 1, pp. 21–29, 2015.

[28] 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.

[29] C. Gaz, F. Flacco, and A. De Luca, “Extracting feasible robot parameters from dynamic coefficients using nonlinear optimization methods,” in IEEE International Conference on Robotics and Automation, 2016, pp. 2075–2081.

[30] J. Bobrow, “Optimal Robot Path Planning Using the Minimum-Time Criterion,” IEEE Journal of Robotics and Automation, vol. 4, no. 4, pp. 443–450, 1988.

[31] O. Chocron and P. Bidaud, “Genetic design of 3D modular manipulators,” in IEEE International Conference on Robotics and Automation, 1997, pp. 223–228.

[32] M. Althoff, M. Mayer, and R. Muller, “Automatic synthesis of human motion from temporal logic specifications,” in IEEE International Conference on Intelligent Robots and Systems, 2020, pp. 4040–4046.