Science Robotics

AAAS

Effortless creation of safe robots from modules through self-programming and self-verification

M. Althoff[1]*†, A. Giusti[1,2]†, S. B. Liu[1]†, A. Pereira[1,3]†

[1]Cyber-Physical Systems Group, Technical University of Munich, 85748 Garching, Germany.

[2]Team of Automation and Mechatronics, Fraunhofer Italia Research, Bolzano 39100, Italy.

[3]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany.

*Corresponding author. E-Mail: althoff@tum.de
† All authors contributed equally to this paper

Abstract Industrial robots cannot be reconfigured to optimally fulfill a given task and often have to be caged to guarantee human safety. Consequently, production processes are meticulously planned so that they last for long periods to make automation affordable. However, the ongoing trend toward mass customization and small-scale manufacturing requires purchasing new robots on a regular basis to cope with frequently changing production. Modular robots are a natural answer: Robots composed of standardized modules can be easily reassembled for new tasks, can be quickly repaired by exchanging broken modules, and are cost-effective by mass-producing standard modules usable for a large variety of robot types. Despite these advantages, modular robots have not yet left research laboratories because an expert must reprogram each new robot after assembly, rendering reassembly impractical. This work presents our set of interconnectable modules (IMPROV), which programs and verifies the safety of assembled robots themselves. Experiments show that IMPROV robots retained the same control performance as nonmodular robots, despite their reconfigurability. With respect to human-robot coexistence, our user study shows a reduction of robot idle time by 36% without compromising on safety using our self-verification concept compared with current safety standards. We believe that by using self-programming and self-verification, modular robots can transform current automation practices.

Summary We effortlessly create and safely operate industrial robots from modules through self-programming and self-verification.

INTRODUCTION

Robotics has transformed the manufacturing sector in recent years. However, small and medium-sized enterprises have not yet reaped the benefits. Reasons for this are manifold: (i) Experts for programming and ensuring correct functionality of robotic solutions are lacking. (ii) Purchasing specialized robots is uneconomical for small-scale manufacturing and/or reacting flexibly to changing demands. Although the concept of physically assembling different robots from a collection of modules is not new (*1*), each new robot composition must be programmed by experts, which is time-

consuming and expensive. (iii) Many production processes require that humans and robots work closely together; ensuring safe human-robot coexistence today requires a large separation of humans and robots (*2*, *3*), making state-of-the-art solutions impractical. Still, 13,000 injuries and 60 deaths were caused by contact with machinery between 2014 and 2018 in the United Kingdom alone (*4*). If machines were able to verify each occurring situation by self-verification, i.e., only carry out safe actions in any given situation, then human error could be avoided.

We propose to untangle the abovementioned issues in a holistic way through the concept of interconnectable modules for self-programming and self-verification (IMPROV), providing three major improvements:

- **Self-programming.** After assembling standardized modules, the created robot programmed itself based on standardized information stored in each module (see Fig. 1A). This allowed us to provide out-of-the-box functionality for a given reference trajectory by generating a model of the robot on the fly. Self-programming of high-level tasks was not considered in this work. The created models were used for automatically synthesizing model-based controllers, as well as for the following two aspects.
- **Self-verification.** To account for dynamically changing environments, the robot formally verified, by itself, whether any human could be harmed through its planned actions during its operation. A planned motion was verified as safe if none of the possible future movements of surrounding humans leads to a collision, as presented in Fig. 1B. Because uncountable possible future motions of surrounding humans exist, we bound the set of possible motions using reachability analysis as explained in more detail in Results. Our inherently safe approach renders robot cages unnecessary in many applications.
- **Optimal module composition.** We automatically chose the best composition of modules for given robot tasks through optimization (see Fig. 1C). The main goals were to minimize cycle time and to reduce energy consumption of the modular robot.

We believe that IMPROV enables users to swiftly change automation solutions without having to purchase new robots, to reprogram underlying controllers, and to redesign the safety concept; thus, flexibility is particularly improved for automation solutions that are frequently redesigned. Below, we review previous work on ensuring out-of-the-box functionality, human safety, and optimality for robotic solutions.

Ensuring out-of-the-box functionality

Previous approaches for controlling modular robots are decentralized; i.e., each module is controlled independently from other modules. A classical approach in decentralized robotic control is the use of proportional-integral-derivative (PID) control as presented in section 8.3 of (*5*); however, the resulting closed-loop robot dynamics is not necessarily stable. Approaches in (*6*, *7*) enhance simple PID controllers for global stability, but their tuning is nontrivial because the knowledge of norm bounds of model terms is necessary. Given the same hardware, centralized control can always be made to perform equally well or better than decentralized control because a decentralized approach is a special case. Despite the degraded performance of decentralized control, many attempts have been made to mitigate its issues: fuzzy gain tuning (*8*), adaptive decentralized control (*9*, *10*), incorporation of joint torque sensing (*11*), robust control (*12*), and virtual decomposition (*13*, *14*).

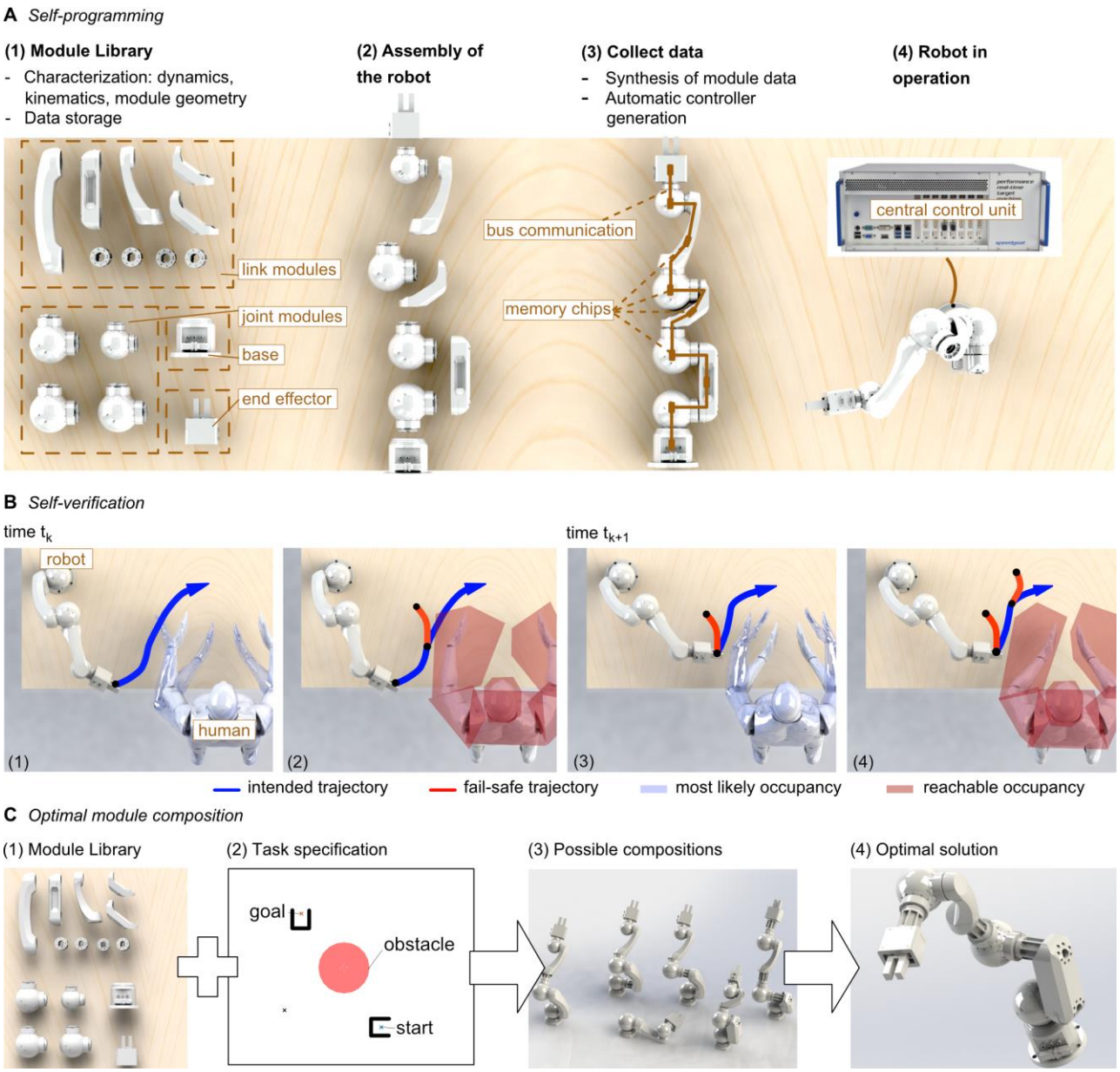

Fig. 1. Self-programming, self-verification, and optimal composition of the robot. (A) Self-programming before deployment: (1) Each module contains its own characterization. (2) The robot is assembled from standardized modules. (3) The information from each module is collected to automatically build a kinematic, dynamic, and geometric model. (4) The robot fulfills a given task without programming by a user. **(B)** Self-verification during deployment: (1) An intended trajectory is planned on the basis of the most likely movement of the human. (2) A fail-safe trajectory is created such that the robot never intersects the reachable set predicted until the robot reaches the end of the fail-safe trajectory. (3) At the next point in time t_{k+1}, the robot updates its intended trajectory. (4) In case the updated intended trajectory combined with a new fail-safe trajectory can be verified on time, the updated intended trajectory is executed; otherwise, the fail-safe trajectory is engaged. **(C)** Optimal module composition before deployment: Given (1) modules and (2) a task, we (3) compute possible compositions and (4) select the optimal composition.

To overcome the shortcomings of decentralized control, our self-programming concept automatically generates a kinematic, dynamic, and geometric model of a newly assembled robot from modules. This not only benefits the motion control of the robot but also is needed for the self-verification concept presented later. So far, only isolated solutions for (semi-)automatic model generation have been developed. One of the seminal works in modular robotics using data stored in modules for obtaining the gravity vector of a robot model is (*15*), but it did not present details for automatically obtaining

the dynamics of the assembled arm. Models have been derived in (*16-18*) based on Lie groups and the product-of-exponentials formulation (*19*); however, these methods have not shown seamless applicability to arbitrarily shaped modules (*20*). On the other hand, our approach builds on the standard Denavit-Hartenberg convention (*21*) and is applicable to arbitrary modules. Extensions of the Denavit-Hartenberg notations have been previously presented, but no complete approach for out-of-the-box operation has been presented: In (*22*), only revolute joints are considered; in section 2.8.2 of (*5*), only special cases of consecutive joint axes are presented, and (*20, 23*) have not addressed the problem of non-uniqueness of the Denavit-Hartenberg convention.

Ensuring human safety

To fully use the benefits of modular robots, robots must no longer be caged, thus requiring new methods to ensure safety. Properties of an impact that could cause serious harm to a human were characterized in (*24*). Reactive control methods have been proposed to reduce the time of impact and the risk of clamping (*25*). Other control methods limited forces (*26*) or impact energy of the end effector (*27*); the performance of the robot was then limited to the low speeds that such methods require. Another approach to reducing the impact energy was to build soft robots (*28*). These robots can be made either from rigid links with inherent compliance in the joints or actuators (*29*) or from deformable materials (*30-32*).

Our approach is different: Instead of reducing the effects of impacts, we provide a formal technique that proves collisions are not possible, except when the robot is at rest. Because the exact future behavior of surrounding humans is unknown, we computed the entire set of possible behaviors using reachability analysis (*33-36*). Inevitable collision states are alternative formal techniques (*37, 38*), but they cannot yet be extended to the high-dimensional problem of robotic manipulators.

Alternative approaches for collision avoidance exist, but none can prove the absence of collisions, which is particularly important due to the apparent safety risks. A simple concept for collision avoidance is that of potential fields (*39*), where obstacles exert repulsive virtual forces on the manipulator. This concept has been extended in many ways: Flacco et al. (*40*) accounted for the incomplete sensing of the environment and also for the velocity of the obstacle when generating repulsive virtual forces as introduced earlier in (*41*). Safety fields (*42*) created repellent forces that depended on not only the relative position but also the relative velocity of humans and robots. Using quadratic programming, they maximized adherence to the desired task while minimizing this measure of danger (*43*).

Ensuring optimality

Another important challenge for adopting modular robots is determining the optimal combination of modules for given tasks. Very few previous works considered heterogeneous modules and their dynamic properties for the automatic synthesis of modular robots. We do not survey works on identical modules because robots assembled from these are typically not aimed for industrial use (*1, 44, 45*).

In (*46*), an algorithm was presented that enumerated all possible compositions of a robot and eliminated kinematically equivalent assemblies. A minimized degrees-of-freedom approach (*47*) found the task-based optimal composition of modules using the enumeration algorithm in (*46*). A composition synthesis methodology was proposed in (*48*), but only for kinematic task requirements, such as workspace volume or maximum reach, and not for dynamic task requirements, such as maximum payload or maximum joint velocities. To reduce the search space, Farritor et al. (*49*) presented a hierarchical synthesis approach, which grouped useful combinations of basic modules;

however, no dynamic task requirements were considered. Composition synthesis approaches based on genetic algorithms were presented in (*49-53*), also without dynamic task requirements.

In contrast to previous works, we consider heterogeneous modules, dynamic task requirements, and an individual solution for each possible composition and present a computationally efficient composition synthesis algorithm, which eliminates failed compositions step by step.

RESULTS

Following Fig. 1, we first present our results for self-programming and then our self-verification procedure. Last, we present how we obtained optimal compositions for a given task.

Self-programming of the robot

A key concept of our approach is that the assembled robot is aware of the parts it consists of. For this reason, we stored the information characterizing each module in itself as illustrated in Fig. 1A. After assembling the robot, the central control unit collected the module data to build a kinematic, dynamic, and geometric model of the assembled robot. Last, the control unit automatically generated code, so that the motion control worked out of the box after assembly. This work considers serial kinematics and builds on our previous work in (*54*) for controller synthesis; we extended it for collision checking, self-verification, and composition synthesis in this work. Because of the previous work in this area, we kept this section concise, and refer to the Supplementary Materials for detailed information.

A module is considered to be a rigid object that serves as a building block for composing a serial robot arm by means of standardized connectors. We distinguished between joint modules and link modules as shown in Fig. 1A, whose schematic illustration is shown in Fig. 2; joint modules add degrees of freedom to the robot, and link modules do not add degrees of freedom. The two parts of a joint module connected by a joint are referred to as the proximal and distal part, as illustrated in Fig. 2. After assembly, we grouped all modules connecting two subsequent joints and refer to this group as an arm link. Note that one joint module can have more than one degree of freedom.

One typically derives the forward kinematics for serial robot manipulators by multiplying the homogeneous transformation matrices relating consecutive reference frames fixed at arm links from the base to the end effector (*5*). The Denavit-Hartenberg convention provides a systematic method for describing these link-fixed frames (*21*); however, this convention is not unique, rendering an automatic computation of the kinematics after assembly impossible. To ensure a nonambiguous solution, we added additional conventions and two parameters (p_i and n_i) to resolve ambiguity as shown in Fig. 2:

- When two consecutive z axes intersect, the x_i unit vector is obtained from their cross product.
- When two consecutive z axes are parallel, the x_i unit vector is set along the common normal between them, and the origin o_i is set at the joint connection PJ_i.
- When two consecutive z axes overlap, the x_i unit vector is aligned with x_{i-1}, and the origin o_i is set at the joint connection PJ_i.

The parameter p_i is the z coordinate of the point PJ_{i-1} measured from $o_{i'}$, and the parameter n_i is the z coordinate of the point PJ_i measured from o_i. Using these values, the previously ambiguous values d_i and θ_i are now uniquely defined as

$$d_i = \begin{cases} n_{i-1} - p_i, & \text{(revolute joint)} \\ n_{i-1} - p_i + q_i, & \text{(prismatic joint)} \end{cases}, \qquad \theta_i = \begin{cases} \gamma_i + q_i, & \text{(revolute joint)} \\ \gamma_i, & \text{(prismatic joint)} \end{cases},$$

where γ_i is an angular offset between consecutive x axes when the joint angle q_i is zero.

To obtain the extended DH parameters (a_i , α_i , γ_i , p_i , and n_i) of the assembled robot automatically, the central control unit gathers kinematic data between joint connections (i.e., for the generic link i , from PJ_{i-1} to PJ_i as shown in Fig. 2). For generating the geometric model for collision checking of the assembled robot, we collected the geometry of each module and applied the same homogeneous transformations as for the kinematic model. When an object is grasped, the shape of the grasped object is added to the collision model. The object geometry may be known beforehand or can be reconstructed from sensor data. Note that our extended Denavit-Hartenberg convention is compatible with the standard convention so that our approaches can also be applied to existing robot platforms.

Next, we derived the dynamic model, which requires some notation. Superscripts of matrices and vectors identify in which frame they are defined. Input and output frames at the connection interface of modules are denoted by *in* and *out* , respectively, as shown in Fig. 2. When a connection is established, the *in* and *out* frames match, and we denote the common frame by *io* . We demonstrate how to automatically obtain the dynamic parameters of an arm link composed of one or several modules. Obtaining the dynamic model of the fully assembled robot from arm links is well known (*5*).

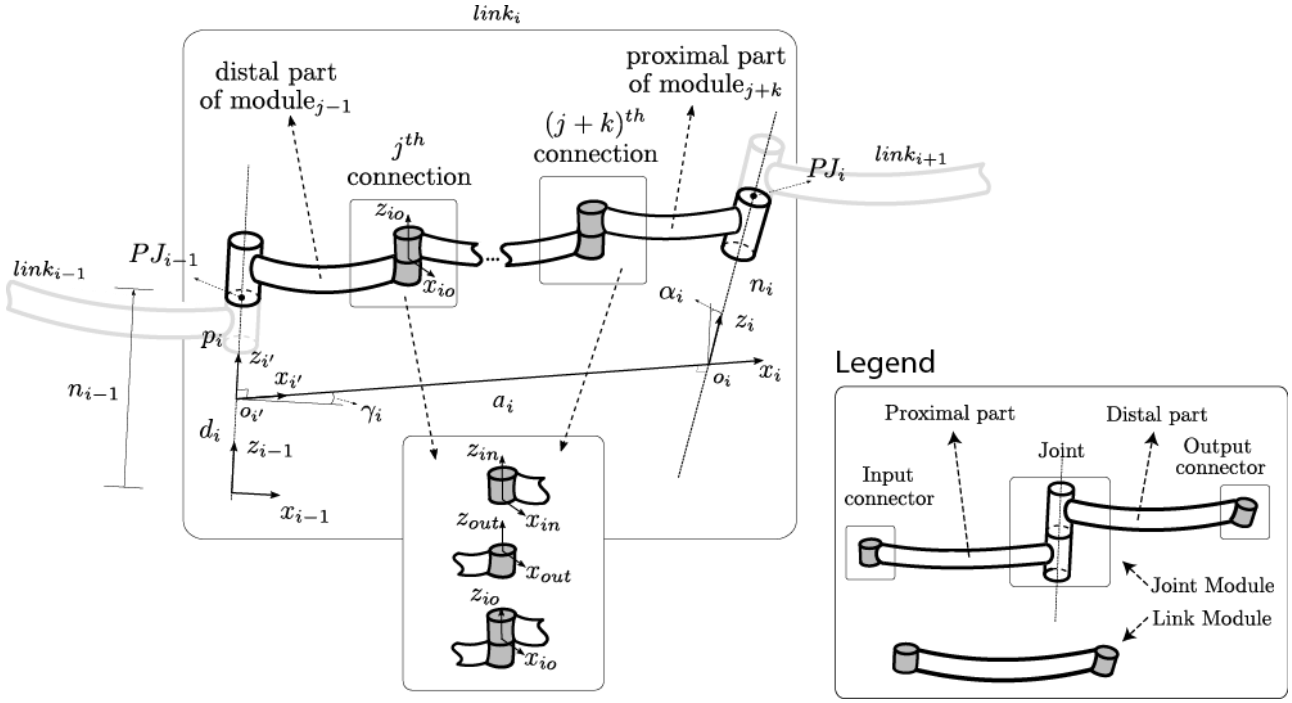

Fig. 2. Extended Denavit-Hartenberg notation for the ith arm link of a robot assembled from modules. Fixed connections are shown in gray, and rotating connections (i.e., joints) are shown in white. The distal part of joint module *j-1*, *k*-link modules, and the proximal part of module *j+k* constitute the ith arm link of the robot.

To obtain the dynamic model of the assembled arm, we used the recursive Newton-Euler (N-E) algorithm (*55-57*), which requires the dynamical parameters of each arm link. Let us introduce for the i^{th} arm link the mass m_i, its inertia tensor $\mathbf{I}_i$, and its center of mass $\mathbf{r}_i$. We first consider the

connection of a link module to a joint module. Using basic mechanics of rigid bodies, the dynamical parameters of the rigid body resulting from this auxiliary connection are

$$m_{a,j} = m_{dl,j-1} + m_{l,j}, \quad \mathbf{I}^{io}_{a,j} = \mathbf{I}^{out}_{dl,j-1} + \mathbf{I}^{in}_{l,j}, \quad \mathbf{r}^{io}_{a,j} = \frac{m_{dl,j-1}\mathbf{r}^{out}_{dl,j-1} + m_{l,j}\mathbf{r}^{in}_{l,j}}{m_{a,j}},$$

where *dl* refers to the distal part of the joint module, *l* refers to the link module, and a refers to the auxiliary connection (see Fig. 2). The inertia tensor expressed in the output frame using Steiner's theorem is

$$\mathbf{I}^{out}_{a,j} = (\mathbf{R}^{io}_{out,a,j})^T \left(\mathbf{I}^{io}_{a,j} - m_{a,j}\mathbf{S}^T(\mathbf{r}^{io}_{a,j})\mathbf{S}(\mathbf{r}^{io}_{a,j})\right)\mathbf{R}^{io}_{out,a,j} + m_{a,j}\mathbf{S}^T(\mathbf{r}^{out}_{a,j})\mathbf{S}(\mathbf{r}^{out}_{a,j}),$$

where $\mathbf{S}(\alpha)$ returns a skew-symmetric matrix so that the multiplication with a vector β equals the cross-product ($\mathbf{S}(\alpha)\beta = \alpha \times \beta$) and $\mathbf{R}^c_d$ denotes a rotation matrix of frame d with respect to frame c. When another link module is added, IMPROV applies the above operations analogously. The last connection to be considered for an arm link is between the auxiliary distal part described above and the proximal part. This last connection establishes the final parameters of the arm link similarly to $m_{a,j}$ and $\mathbf{I}^{out}_{a,j}$:

$$m_i = m^{dl}_{j-1+k} + m^{pl}_{j+k}, \quad \mathbf{I}^{io}_i = \mathbf{I}^{out}_{dl,j-1+k} + \mathbf{I}^{in}_{pl,j+k}, \quad \mathbf{r}^{io}_i = \frac{m^{dl}_{j-1+k}\mathbf{r}^{out}_{dl_{j-1+k}} + m^{pl}_{j+k}\mathbf{r}^{in}_{pl_{j+k}}}{m_i},$$

and

$$\mathbf{I}_i = (\mathbf{R}^{io}_i)^T \left(\mathbf{I}^{io}_i - m_i\mathbf{S}^T(\mathbf{r}^{io}_i)\mathbf{S}(\mathbf{r}^{io}_i)\right)\mathbf{R}^{io}_i.$$

To obtain the dynamical model, IMPROV uses the obtained data for kinematics and dynamics of all arm links; the Denavit-Hartenberg parameters are stored in the table **DH**, and the dynamic parameters are stored in the table **DynPar**. After introducing $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ as the vectors of joint positions, velocities, and accelerations, respectively, we obtain the dynamical model using the recursive N-E algorithm $NEA_g(\cdot)$ (*55*):

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = NEA_g\left(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}},\mathbf{DH},\mathbf{DynPar}\right).$$

We have implemented several motion control concepts using the automatically derived dynamic model. The following controllers can be automatically implemented by IMPROV: inverse-dynamics control [see section 8.5.2 of (*5*)], passivity-based control [see, e.g., section 8.4 of (*58*)], and passivity-based control with adaptive friction compensation based on (*59*). To compute a reference trajectory in joint space from a desired trajectory in task space, we used the automatically generated kinematic model. We solved a second-order inverse kinematics scheme with quaternion feedback based on (*60*) by a damped least-squares approach (*61*) to circumvent singularity issues. In addition, we damped floating null-space motions (*62*) arising from redundant robot assemblies as shown in (*63*).

For validating the accuracy of the automatically obtained models and the performance of the controllers, we have conducted several experiments whose details and plots are presented in the Supplementary Materials. In all cases, the same performance levels could be replicated even after changing the composition. Because of the self-programming of modern control concepts, the performance was superior to a classical PID control scheme for both tracking performance and stability.

Self-verification of the robot

Because various robots are assembled from IMPROV, it is not feasible to verify collision avoidance capabilities after each new composition. Furthermore, future motions of humans in the vicinity of the robot are unknown. To consider these two aspects, we have developed self-verification capabilities that allow the robot to verify online whether its currently planned action is safe. The robot verifies itself that it must stop before any human can touch it. Next, we explain our concept as it applies to a single human, which one can trivially extend for arbitrarily many humans. To ensure that we capture all possible behaviors of humans, we use reachable sets. After introducing a possible trajectory of a human as $\chi(t;\mathbf{x}(0),\mathbf{u}(\cdot))$ for time t, initial state $\mathbf{x}(0)$, and input trajectory $\mathbf{u}(\cdot)$, we define the set of reachable sets for a set of initial states $\mathcal{X}_0$ and a set of possible input values $\mathcal{U}$ as

$$\mathcal{R}(t) = \{\chi(t;x_0,u(\cdot)) \mid \mathbf{x}(0) \in \mathcal{X}_0, \forall t : \mathbf{u}(t) \in \mathcal{U}\}.$$

The *reachable set* of a time interval $[t_0,t_1]$ is defined as $\mathcal{R}([t_0,t_1]) = \bigcup_{t\in[t_0,t_1]} \mathcal{R}(t)$. For subsequent discussions, we also introduce the area operator $\mathcal{A}(\mathbf{x}(t))$, returning the occupancy of the robot for a given state vector $\mathbf{x}(t)$ and the *reachable occupancy* as

$$\Gamma([t_0,t_1]) = \{\mathcal{A}(\chi(t;x_0,u(t))) \mid \chi(t;\mathbf{x}(0),\mathbf{u}(t)) \in \mathcal{R}(t), t \in [t_0,t_1]\}.$$

Our self-verification concept uses the principle of induction. We first present the base case (prove that the specification holds initially) followed by the inductive step (prove that, if the statement holds for step k, then the statement holds for k + 1).

Base case for the first time interval [t_0,t_1]. The robot can only be started if it is initially at rest. Before time t_0, we plan an intended trajectory, plan a fail-safe maneuver one time interval into this intended trajectory (i.e., at time t_1), and verify that the robot does not intersect the reachable occupancy of the human, during neither the first part of the intended trajectory nor the fail-safe trajectory, as presented in Fig. 1B, 1 and 2. The trajectory is only executed at t_0 if this is verified. Because the fail-safe maneuver is constructed so that the robot is stationary at the end, we verify that the robot is not moving when touched by a human.

Inductive step from time interval [t_{k-1},t_k] to [t_k,t_{k+1}]. The next step ([t_k,t_{k+1}]) of the intended trajectory is only executed if it, together with a fail-safe maneuver starting at t_{k+1}, can be verified as safe before t_k (see Fig. 1B4). If so, the intended trajectory is followed at t_k. Otherwise, because the fail-safe maneuver starting at t_k of the previous time interval ([t_{k-1},t_k]) has already been verified in the previous time step, this fail-safe maneuver is executed.

To focus on the interaction between motion planning and reachability analysis, we assumed that a long-term plan for the robot considering the most likely movement of the human exists [see, e.g., (*64-67*)]. To improve efficiency, one can use planners that anticipate the future movement of surrounding humans (*68-70*). The fail-safe motions are computed on much shorter time scales and require very efficient implementations. We used path-consistent fail-safe trajectories because they are expected to be more predictable for the user, and predictability increases trust (*71*).

A remarkable property of our approach is that only the intended trajectory and fail-safe maneuver must be stored in the robot memory. In addition, even if the computer hardware for computing new trajectories breaks down, the robot remains safe because the default path is the fail-safe maneuver. In the event that a fail-safe trajectory is triggered, one need not follow the fail-safe maneuver until the robot is stationary. During the fail-safe trajectory, one can already compute a recovery maneuver branching off a fail-safe maneuver to recover to the original plan.

To ensure that the fail-safe maneuver is rarely engaged, we required very fast motion planning and verification. Considering the motion planner, we used the approach in (*72*); because no fast algorithm for predicting over-approximative human occupancy has been developed elsewhere, we present our approach below. In trials, an update frequency of 500 Hz was found to be practical. To meet these high computational demands, we used two quick-to-compute approaches in parallel, as shown in Fig. 3A. The joint-space approach considers that distances between joints are constant because of the bones of the human body. However, this approach is subject to singularities so that we additionally predict the occupancy in task space. Predictions of both methods are over-approximative; i.e., the obtained predictions account for all possible movements (*73*). Thus, even if only one prediction is verified as safe, the actual movement of the human will not intersect with the occupancy of the robot, and the robot is verified as safe. This parallelization concept is important because none of the presented approaches is superior in all cases. We first present the occupancies for a single human arm and later extend the results for the entire human body.

Self-verification in joint space. Our joint-space approach computes the reachable set of joint angles for a kinematic model of the human and maps these to occupancy in task space. The initial set of joint angles $\mathcal{X}_0$ is obtained by fitting a kinematic model to the human and adding an uncertain set capturing measurement uncertainties (as shown in the upper half of Fig. 3A). To obtain tight over-approximations, we considered joint values, joint velocities, and joint accelerations limited for humans. These constraints can be modeled by hybrid systems (*74*); however, computing reachable sets of hybrid systems is computationally demanding (*35*). Instead, we added moderate conservativeness by computing with models obtained from basic kinematics for maximum joint angles, joint velocities, and joint accelerations in parallel and then intersecting the results:

- Zeroth-order model of maximum joint position:
  $\mathcal{R}_q^{(1)} = [\mathbf{q}_{inf}, \mathbf{q}_{sup}]$,
- First-order model of maximum joint velocity (Minkowski sum:
  $\mathcal{A} \oplus \mathcal{B} = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\}$):
  $\mathcal{R}_q^{(2)}(t) = \mathcal{Q}(0) \oplus [\dot{\mathbf{q}}_{inf}, \dot{\mathbf{q}}_{sup}]t$,
- Second-order model of maximum joint accelerations [conv() returns the convex hull]:
  $$\mathcal{R}_q^{(3)}(t) = \mathrm{conv}\left(\mathcal{Q}(0), \mathcal{Q}(0) \oplus \dot{\mathcal{Q}}(0)t \oplus \left[\ddot{\mathbf{q}}_{inf}, \ddot{\mathbf{q}}_{sup}\right]\frac{t^2}{2}\right).$$

The input bounds $\mathbf{q}_{inf}$, $\mathbf{q}_{sup}$, $\dot{\mathbf{q}}_{inf}$, $\dot{\mathbf{q}}_{sup}$, $\ddot{\mathbf{q}}_{inf}$, and $\ddot{\mathbf{q}}_{sup}$ have been validated using an established database of human motion and a high-fidelity simulator, as described later. Intersecting the partial results returns an over-approximation of the reachable set:

$$\mathcal{R}_q(t) = \bigcap_{i=1}^{3} \mathcal{R}_q^{(i)}(t).$$

On the basis of the reachable set in joint space $\mathcal{R}_q(t)$, we obtain the reachable occupancy in task space $\Gamma(t)$ using sphere-swept volumes (*75*). A sphere-swept volume is the Minkowski sum of a convex hull of a set of points $\mathcal{P} = \{P_1, P_2, \ldots, P_l\}$ and a ball of radius r, denoted by $\mathcal{B}(r)$: $\mathcal{V} = \mathrm{conv}(\mathcal{P}) \oplus \mathcal{B}(r)$.

For the task-space approach described below, we also require capsules, which are a special case of a sphere-swept volume, where $\mathrm{conv}(\mathcal{P})$ is a line segment. At singularities of the human arm (*5*), we only use the self-verification in task space.

Self-verification in task space. The task-space approach computes the reachable occupancies directly in the task space as sets of capsules, avoiding problems with singularities in the kinematic models. The intersection of capsules is computationally expensive. However, capsule-to-capsule intersection checks are fast, as presented in section 4.5 of (*76*), which we exploit by only checking for intersection with the robot for each model: If one model does not cause a collision, the trajectory of the robot is still safe.

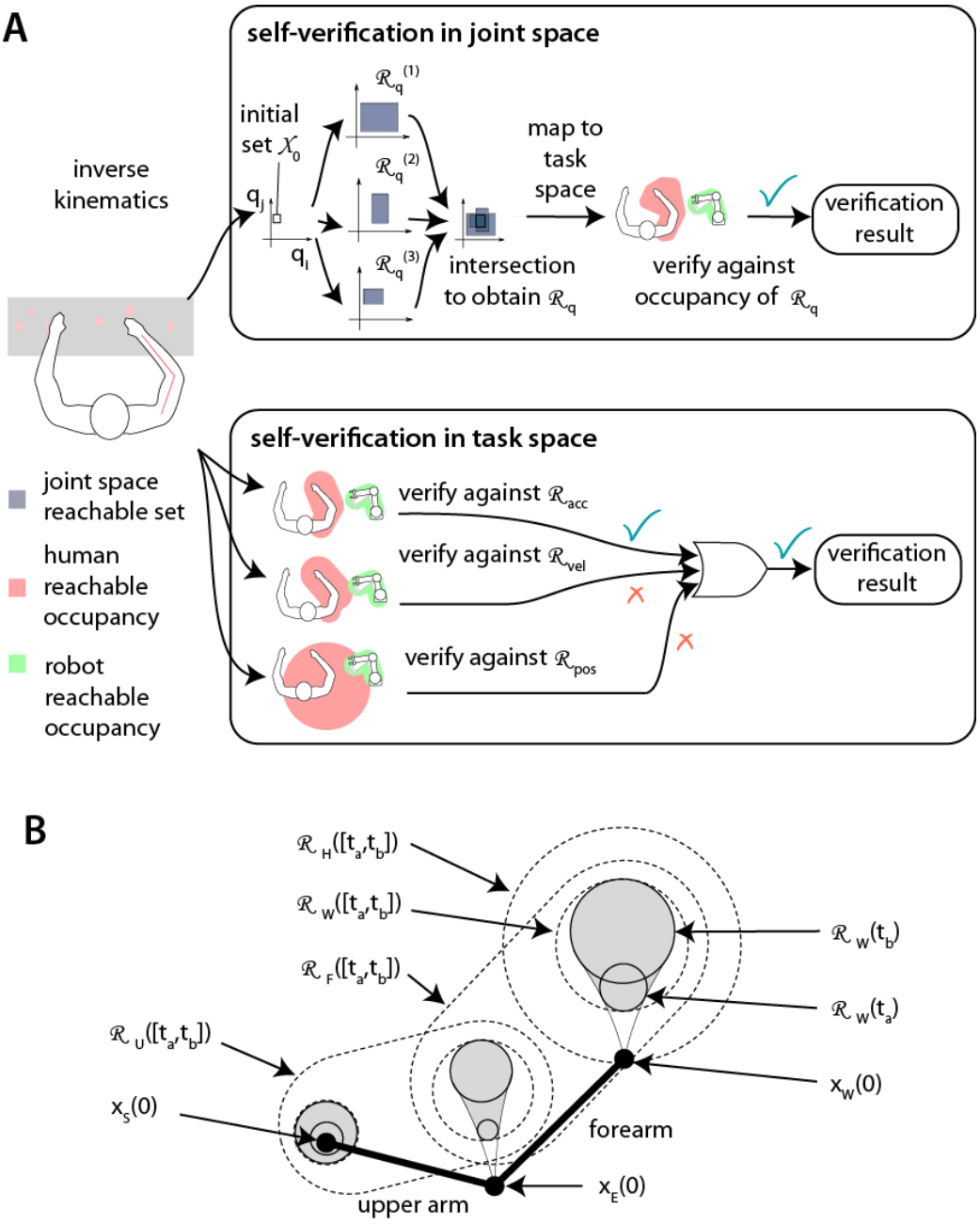

Fig. 3. Online verification using reachable occupancies. (A) Two verification concepts are used in parallel: The upper part shows the verification procedure in joint space, and the lower one directly computes results in the task space. **(B)** Occupancy of the human arm using the task-space method.

After introducing the initial position of a point on the human body as $y(0)$, its measurement uncertainties $\delta\mathbf{y}$ and $\delta\dot{\mathbf{y}}$ for position and velocity, respectively, and its maximum velocity $v_{y,max}$ and

acceleration $a_{y,max}$, the reachable position of a point mass using a first- and second-order model can be bounded using basic kinematics by

$$\mathcal{R}_y^{(1)}(t) = \mathbf{y}(0) \oplus B(\delta y) \oplus B(v_{y,max} \cdot t),$$

$$\mathcal{R}_y^{(2)}(t) = \mathbf{y}(0) \oplus B(\delta y) \oplus \dot{\mathbf{y}}(0) \cdot t \oplus B(\delta \dot{y} \cdot t) \oplus B\left(\frac{a_{y,\max}}{2} \cdot t^2 \right).$$

The reachable set of a single point on the arm is used to assemble capsules enclosing possible future occupancies of the entire arm as shown in Fig. 3B. The reachable sets of the i^{th} order model of the shoulder $\mathcal{R}_S^{(i)}(t)$, the elbow $\mathcal{R}_E^{(i)}(t)$, and the wrist $\mathcal{R}_W^{(i)}(t)$ are obtained using $\mathcal{R}_y^{(1)}(t)$ and $\mathcal{R}_y^{(2)}(t)$. We further require the radius r_s of the capsule enclosing the upper arm and forearm. The forearm, upper arm, and hand are enclosed by the following capsules:

- forearm: $\mathcal{R}_F(t) = \text{conv}(\mathcal{R}_E(t), \mathcal{R}_W(t)) \oplus B(r_S)$,
- upper arm: $\mathcal{R}_U(t) = \text{conv}(\mathcal{R}_S(t), \mathcal{R}_E(t)) \oplus B(r_S)$,
- hand: $\mathcal{R}_H(t) = \mathcal{R}_W(t) \oplus B(r_H)$.

Combining the occupancies of different body parts results in the overall occupancy

$$\mathcal{R}_{acc}(t) = \mathcal{R}_F^{(2)}(t) \cup \mathcal{R}_U^{(2)}(t) \cup \mathcal{R}_H^{(2)}(t), \qquad \mathcal{R}_{vel}(t) = \mathcal{R}_F^{(1)}(t) \cup \mathcal{R}_U^{(1)}(t) \cup \mathcal{R}_H^{(1)}(t).$$

The occupancy of the zeroth-order model is simply a ball whose radius is that of an outstretched arm and whose center expands with the maximum velocity of the shoulder using the notations from Fig. 3B:

$$\mathcal{R}_{pos} = \mathbf{x}_S(0) \oplus B(v_{S,\max} \cdot t + \| \mathbf{x}_S(0) - \mathbf{x}_E(0) \| + \| \mathbf{x}_E(0) - \mathbf{x}_W(0) \| + \delta y + r_H).$$

Because we directly obtain the reachable sets in the task space, the reachable set equals the occupancy [$\mathcal{R}(t) = \Gamma(t)$] so that we do not have to transform the reachable states as presented for the joint-space approach.

The prediction of the full body is done analogously, with the only difference that we do not use the joint-space approach for the other body parts. The reason is that the arms are typically more important compared with the other body parts when working close to a robot, so it suffices to use just the task-space approach for the rest of the body.

Example. We demonstrate the self-verification in Fig. 4. Let us introduce the scaling parameter c, which is the ratio between actual and planned velocity and thus ranges between zero and one. As long as the human is sufficiently far away (until $t = t_A$), all trajectories are verified as safe, and the robot runs at full speed (scaling parameter $c = 1$). At $t = t_B$, the human is approaching the robot fast so that the self-verification ("verified?"-signal turns to "no") initiates a fail-safe maneuver. While stopping, the robot repeatedly tries to recover the original trajectory, which is indicated by the verification signal chattering between "yes" and "no". The recovery/fail-safe maneuvers are planned such that the jerk (derivative of acceleration) and acceleration at each joint are bounded and that acceleration, velocity, and position are continuous at the transition from fail-safe to recovery maneuver and vice versa. In practice, this results in the robot organically finding a safe equilibrium speed. At $t = t_C$, the hand is in close proximity to the robot, not allowing the robot to move at all ($c = 0$). Once the human is moving away ($t = t_D$), safe-verified recovery trajectories are found, and the robot starts to move again. A similar scenario is also demonstrated in movie S1, where one can also see the self-verification for a reconfigured robot. The computation time for both the computation of the human

reachable occupancies and their verification against the intended trajectory and fail-safe maneuver is typically around 124 µs for the joint-space approach and 4 µs for the task-space approach (not including image processing from sensors and communication delays).

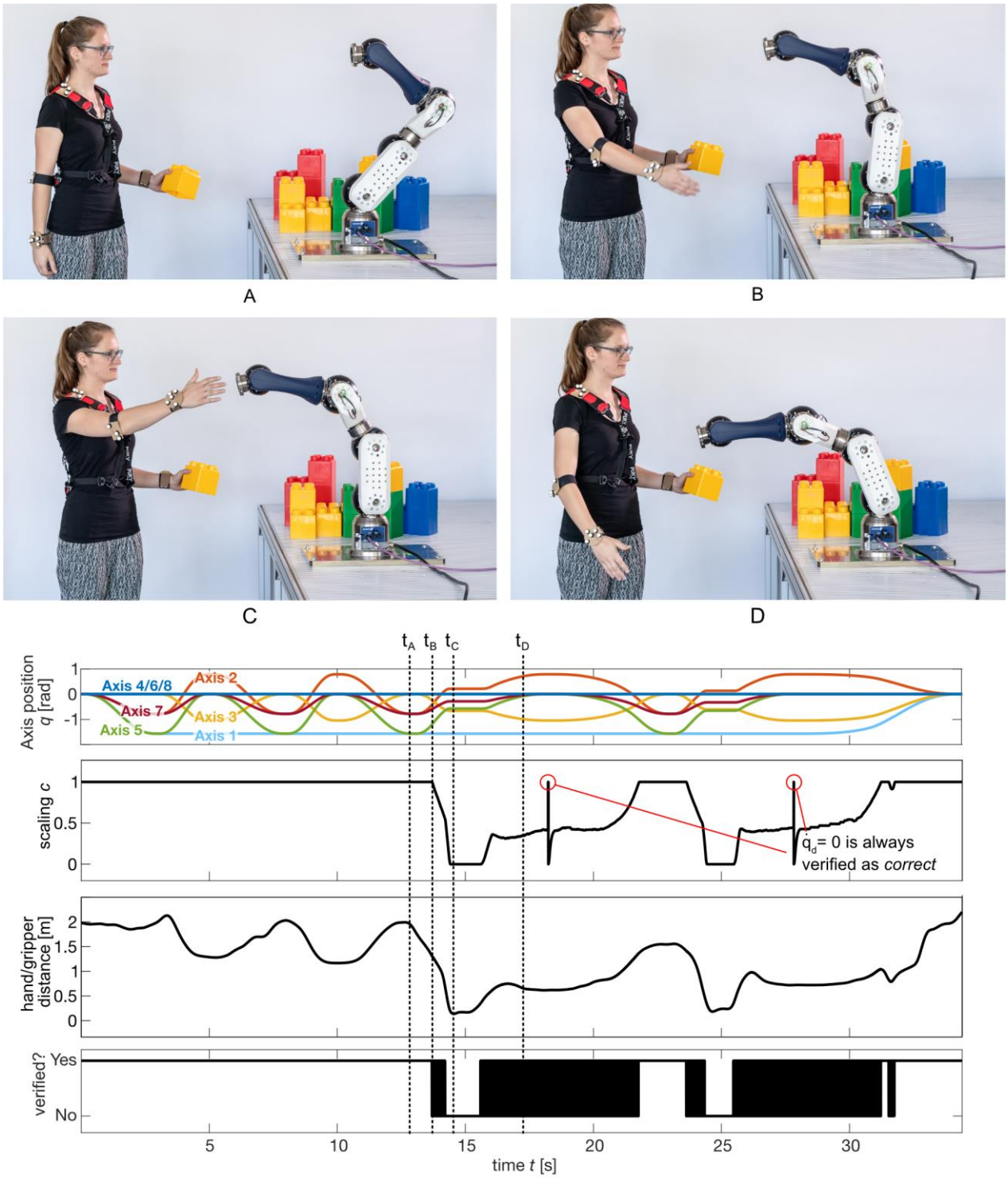

Fig. 4. Self-verification of the robot. (A) time t_A, **(B)** time t_B, **(C)** time t_C, and **(D)** time t_D. The graphs show the axes positions, the trajectory scaling, the distances between the hand and the gripper, and the verification results. Snapshots of four interesting points in time are shown above and are indicated on the graphs below with dashed vertical lines.

User study

To demonstrate the usefulness of self-verification compared with static safety regions, we performed a user study in which participants worked alongside the robot. We chose 30 healthy individuals (14 male and 16 female) aged between 22 and 30 years old. The task was to assemble a simple jigsaw puzzle on a table outside of the robot workspace, whereas the pieces for the puzzle were in the robot workspace as shown in Fig. 5. None of the participants had worked with the robot of the user study before. To account for accustomization to the robot (*77*, *78*), participants had three sessions with the robot, spread over the course of up to 8 days; during each session, the participants completed the task alongside the moving robot four times. To focus on the benefits of self-verification, we did not reassemble the robot in between the trials.

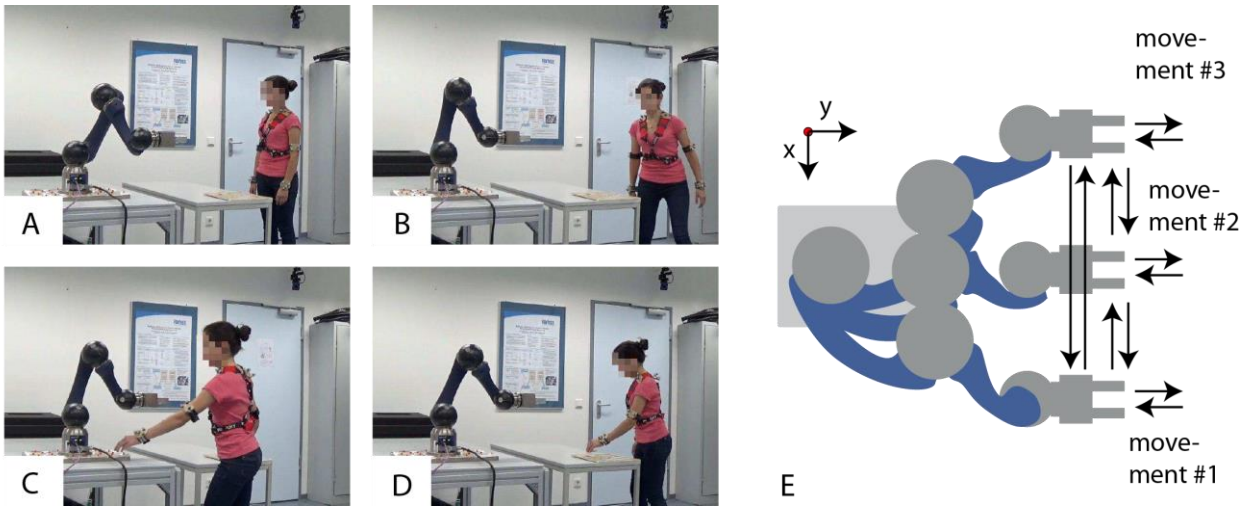

Fig. 5. Setup of the user study. (A) Beginning of the task. **(B)** The participant moves toward the robot to fetch a piece of the puzzle. **(C)** The participant picks a piece from the puzzle. **(D)** The participant places the piece on the other table. **(E)** Template movements of the robot: Away from the base in the y direction for 1.7 s and back again for 1.7 s, at three different starting positions, between which the robot moves randomly.

We compared our method with the safety-rated monitored stop from International Organization for Standardization (ISO) 13855 (*2*) and ISO 10218-1 (*3*), where the robot stops when the human enters the workspace of the robot; details are described in Materials and Methods. The participants were assigned at random to either the implementation using self-verification or the implementation using the static safety zone; the participants did not know which group they belonged to.

Whereas the humans were given a specific task, the goal for the robot was to follow its tasks with as little idle time as possible. Both human time to completion and idle time of the robot improved as the human became accustomed to the robot. In each case, the idle time of the robot was significantly lower using self-verification than with static safety regions, as shown in Fig. 6. The percentage reduction in idle time is 38% ($p < 10^{-5}$; the p-value is the probability that we mistakenly reject the null hypothesis) for the first four trials and 37% ($p < 10^{-5}$) for the last trials; for all trials, it was 36%.

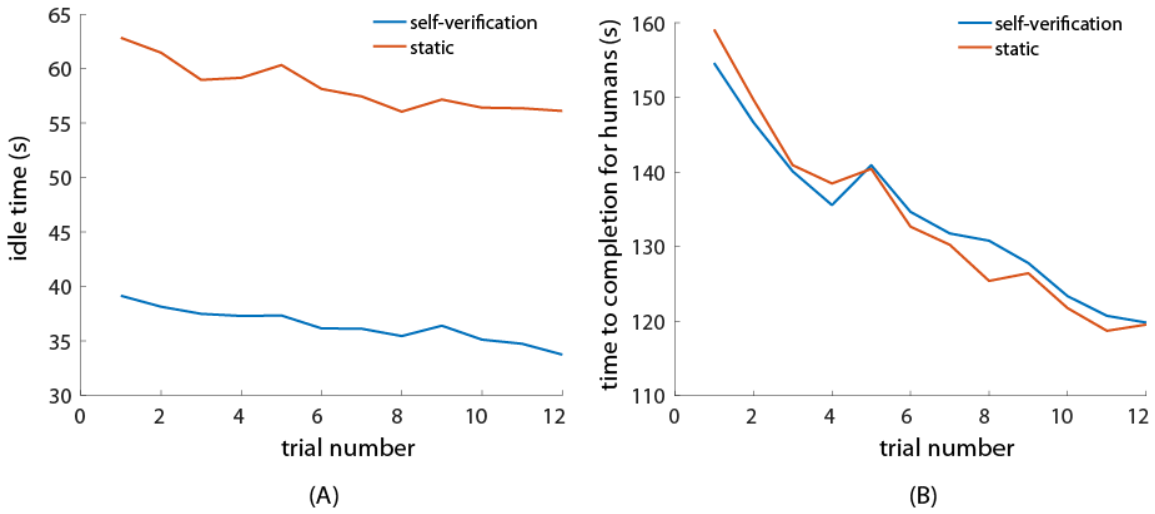

Fig. 6. Results of the user study. (A) The idle time of the robot is significantly reduced when using our self-verification scheme. The idle time remains rather constant over the different trials. **(B)** The time to completion for humans is not affected by the safety concept. It can be seen that humans became faster at solving the jigsaw puzzle due to training effects.

Determining optimal compositions

To fully exploit the potential of modular robots, users are interested in finding the best composition of modules to fulfill a desired task. A selection of our modules and a possible composition is shown in Fig. 7A. As an example, we consider a pick-and-place task, where the robot end effector should move as fast as possible from start (position S) to goal (position G) while avoiding surrounding obstacles. Our implementation returns to the user the order in which certain modules have to be assembled to obtain the optimal composition.

Let us define the set of compositions as $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$, where C_i is the i^{th} composition and N is the number of considered compositions. Our cost function for finding the optimal composition considers the duration $t_{f,i}$ to reach the goal position and the required energy consumptions by the motors. We denote the weight for time optimality by w_t and the weight for energy optimality by w_e. To exclude effects from varying energy efficiency of the motors used in different robots, we assume that no energy losses occur in the motors. After introducing the motor torque vector $\mathbf{u}_i(t)$, the joint velocity vector $\dot{\mathbf{q}}_i(t)$, and the time to reach the goal $t_{f,i}$, all for composition C_i, we can formulate the optimization problem as:

$$\min_{C_i \in \mathcal{C}} w_t t_{f,i} + w_e E_i, \qquad E_i = \int_0^{t_{f,i}} |\mathbf{u}_i(t)|^T |\dot{\mathbf{q}}_i(t)| \, dt$$

subject to $\forall t \in [0, t_{f,i}]$

$\mathbf{q}_i(t) \in [\underline{\mathbf{q}}_i, \overline{\mathbf{q}}_i]$ (joint limits), (1)

$\mathbf{u}_i(t) \in [\underline{\mathbf{u}}_i, \overline{\mathbf{u}}_i]$ (torque limits), (2)

$\mathcal{A}_i(\mathbf{q}_i(t)) \subseteq \mathcal{F}$ (composition C_i stays within the obstacle-free space $\mathcal{F}$), (3)

where $\mathcal{A}_i(\mathbf{q}_i(t))$ returns the occupancy of the robot as introduced before and $\mathcal{F}$ is the space free of obstacles. The torques for each composition are evaluated using our self-programming approach and the N-E algorithm (*55*).

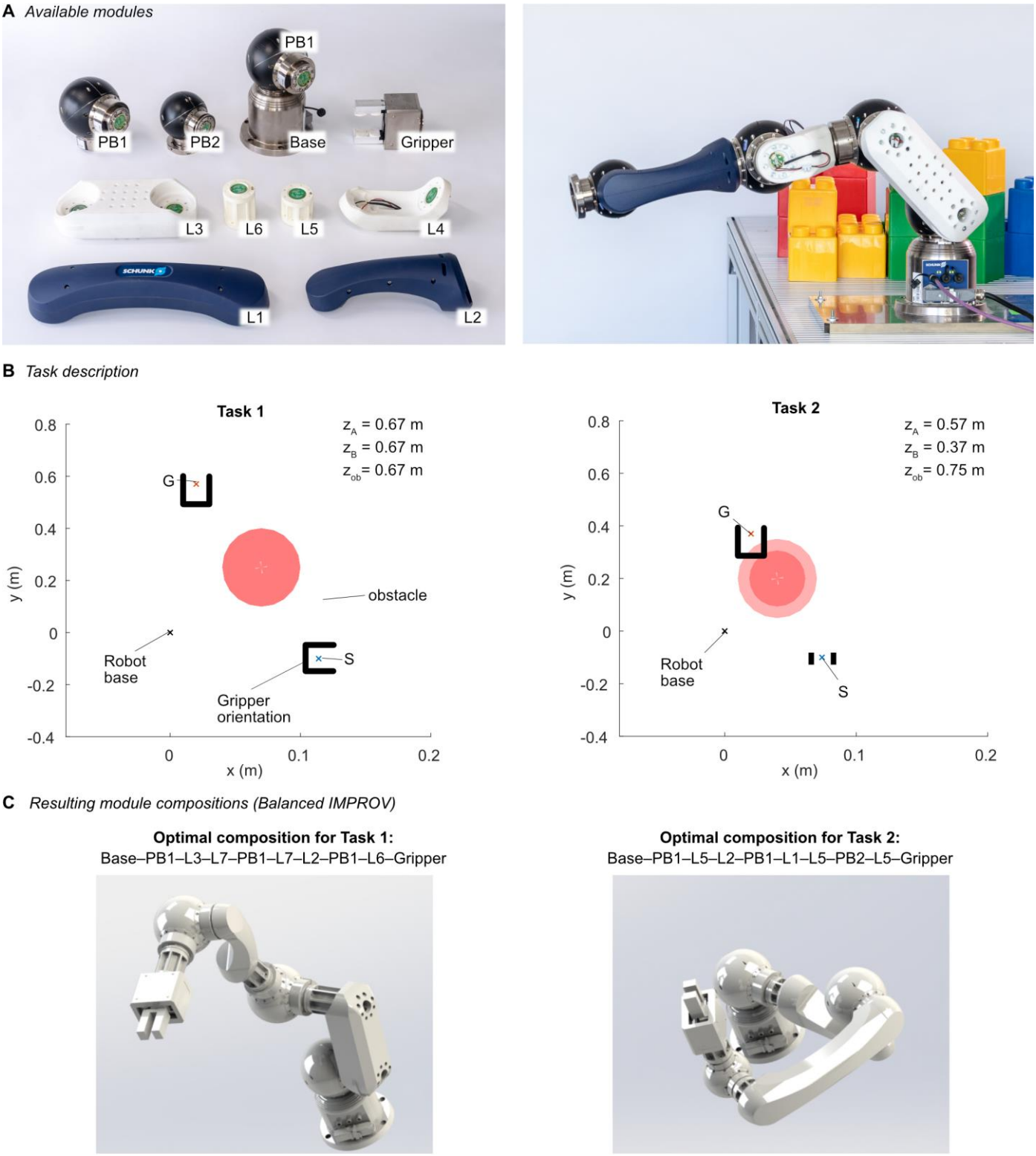

Fig. 7. Determining the optimal module composition out of available modules. **(A)** Available modules and a possible composition. **(B)** Task description of tasks 1 and 2, which consists of start and goal points (S and G), the desired gripper orientation, and obstacles. **(C)** Optimal compositions for tasks 1 and 2.

A brute force approach (*46*) computing the cost function over all possible compositions is subject to the curse of dimensionality because the number of compositions grows exponentially with the number

of modules. In this work, we mitigated this problem by reducing the search space and through step-by-step elimination of assemblies that could not fulfill the given task.

Search space reduction. We reduced the search space by restricting certain combinations of modules; e.g., it does not make sense to first combine all joint modules and then all link modules. To alternate between joint and link modules in a reasonable manner, we restricted the number of link modules in between joint modules by three. In addition, we did not allow connections of joint modules solely established by link extensions because this typically results in distances between joints that are too small and thus limits the freedom of movement of each joint. Last, for a fair comparison with the other six-degree-of-freedom robots, we restricted the degrees of freedom of the assembled robot to six.

Step-by-step elimination. To reduce computation time, we discarded infeasible assemblies by performing simple satisfiability checks of constraints (1)-(3) first, followed by more computationally expensive satisfiability checks. As a consequence, the more complicated checks only have to be performed on the remaining feasible compositions. We performed feasibility tests in the following order:

1. Can the composition reach the start and goal positions (S and G in Fig. 7B)?
2. Are joint and torque limits met at the start and goal positions? This is checked by applying constraints (1) and (2) to positions S and G.
3. Are joint and torque limits met between the start and goal positions? This is checked by applying constraints (1) and (2) to the entire trajectory.
4. Does a collision-free path between the start and goal composition exist (including self-collision)? This is checked by applying constraint (3) to the entire trajectory.

The third and fourth checks require a trajectory of the robot from the start (position S) to the goal (position G). Several techniques for motion planning exist, and all of them can be embedded in our approach. For the study presented here, we computed a trapezoidal trajectory in joint space as presented in section 4.2 of (*5*).

Comparison. We show the advantage of using optimized module compositions for frequently changing tasks by comparing the IMPROV modules (see Fig. 7A) to a standard-configured Schunk LWA 4p, a KUKA LWR 4+, and a Stäubli TX90 in simulation. The robots have to fulfill two different tasks, in each of which the robot has to move from start point S to goal point G while avoiding an obstacle, as shown in Fig. 7B. A spherical obstacle is used because it is easier to understand and reproduce results when using simple geometries; however, our software uses a collision-checking library for arbitrary shapes (*79*). We assumed that the position of the robot base was fixed at $x = 0, y = 0, z = 0$ and that the base could be rotated about the z axis. Furthermore, we assumed $\dot{q}_{max} = 1[rad/s]$ and $\ddot{q}_{max} = 1[rad/s^2]$ for all joints of all robots. To demonstrate how the choice of the cost function affects the solution, we used three different settings:

- Balanced: $w_t = 1,\ w_e = 0.2$
- Time-optimal: $w_t = 1,\ w_e = 0$
- Energy-optimal: $w_t = 0,\ w_e = 1$

Fig. 7C shows the optimal compositions obtained using our approach, and Table 1 compares the results with the previously introduced robots. For task 1, we observed that the robot composed of IMPROV modules avoided the obstacle, whereas trapezoidal trajectories computed in joint space could not return collision-free trajectories for the other robots. For task 2, the KUKA robot is the

fastest but only by a small margin compared with the fastest IMPROV composition, which, in turn, is vastly more energy efficient. Gravity has a large influence on the joint whose movement causes other joints to primarily move in the direction of gravity. For typical industrial robots, this is usually the second joint. In our solution, however, the axis primarily affected by gravity is the fourth axis (see Fig. 7C), so that fewer remaining parts of the robot are strongly affected by gravity. We also observe for task 2 that the IMPROV compositions are better in terms of overall cost. Simulation videos for this comparison are provided in movie S2.

Table 1. Results of comparison between IMPROV modules and other robots. We abbreviate "no solution" by n. s. and "no collision-free trajectory" by n. c. t. Collisions include collisions with other obstacles and self-collisions. The best results in each category are shown in bold.

Robot	Task 1: t_f [s]	Task 1: E_i [J]	Task 1: Cost [-]	Task 2: t_f [s]	Task 2: E_i [J]	Task 2: Cost [-]
Schunk LWA4p	n. s.	n. s.	n. s.	3.38	21.71	7.72
KUKA LWR4+	n. c. t.	n. c. t.	n. c. t.	**2.17**	18.58	5.89
Stäubli TX90	n. c. t.	n. c. t.	n. c. t.	n. c. t.	n. c. t.	n. c. t.
Balanced IMPROV	3.49	30.89	**9.67**	2.23	10.01	**4.23**
Time opt. IMPROV	**3.33**	39.76	11.28	2.21	22.71	6.75
Energy opt. IMPROV	5.70	**23.20**	10.34	2.75	**9.86**	4.72

DISCUSSION

Effortless creation of robotic manipulators can only realize its full potential when combining self-programming and self-verification. When only realizing self-programming, the created robot would have to be caged, so that several benefits of a customized solution are lost: For example, designing a cage for a particular robot is not economical in small-scale manufacturing, which is exactly the targeted application scenario. The second lost benefit is that a customized robot can be optimally integrated into its environment, but a cage would separate the robot from the objects it should manipulate. Other approaches for ensuring safety require demonstrating a sufficiently low risk for the user in case of impact (e.g., considering the power and force limiting of ISO/TS 15066); this, however, requires a dedicated analysis of each different application so that flexibility is lost. We next discuss the advantages of our system regarding programming, conventional safety solutions, and performance compared with standard robots.

Comparison with standard programming approaches

Modular robots are not used in manufacturing today because reprogramming costs exceed installation costs by a large margin (*80*) despite many efforts toward simplifying programming of robots (*81*). This issue becomes even more important when programming robots for safety-critical applications in human-robot coexistence, as discussed in this paper. According to Regan and Hamilton (*82*), around 50 software defects remain in 1000 lines of newly written uncommented code, around 10 defects can be found in thoroughly tested code, and still around 1 defect is found after extreme measures of additional testing. This can lead to hazardous behavior considering the number of lines of codes in software written today. This was well documented during the Defense Advanced Research Projects Agency Robotics Challenge for disaster response applications, where a number of teams failed due

to programming errors despite good engineering practices and extensive testing (*83*, *84*). Thus, self-programming capabilities drastically reduce the probability of incorrectly working software.

Comparison with static safety zones

Our self-verification approach is more efficient compared with static safety zones because, with our approach, the robot only needs to alter its movement if the human is directly in danger of collision, whereas when using static safety zones, the robot stops as soon as the human appears in its workspace. With practice, the human participants became more efficient and faster at their tasks and spent less time in the workspace of the robot as a result, explaining the improvement in time to completion shown in Fig. 6B over time. Because users gain confidence after time due to the formally correct safety verification, self-verification has a lot of potential, due to the significantly reduced idle time compared with static safety zones.

Comparison with non-modular robots

Modular robots have many obvious advantages with respect to flexibility, maintenance, and cost efficiency compared with standard robots. (i) Flexibility: Modular robots can be easily adapted to current needs and enable flexible manufacturing (*44*, *85*, *86*). (ii) Maintenance: Flexible robots are easier to maintain because broken modules can be easily replaced (*1*, *44*). (iii) Cost efficiency: Modular robots are more cost-effective for two reasons. First, one does not require general-purpose machines for flexible manufacturing, creating capital waste; instead, the robots are only assembled to meet their current purpose [see section 2 of (*85*)]. Second, modular robots can be more cost-effective because one only requires a few modules to assemble many different robots and thus modules can be mass-produced (*1*, *44*).

Finding optimal assemblies

Providing custom robotic solutions for a given task can provide substantial advantages. As an illustrative example, consider a pick-and-place task without any obstacles. There always exists a point with the same distance r between the start and the goal positions. By choosing this point as the position of the base, it suffices to have a robot with a single degree of freedom and a link that has length r. Obviously, for environments with obstacles and tasks where the orientation of the end effector is important, this most simple design is not possible. Typically, the number of possible assemblies is too vast to be evaluated by a human. By evaluating tens of thousands of assemblies, one obtains solutions that a human designer—with preconceptions of how a robot should look—might not have considered. An example of such a robot is the one found for task 2 in Fig. 7C, which has a nonstandard kinematics that is not available for purchase. This shows us that there is an untapped potential for nonstandard kinematics, particularly for special tasks and obstacle-laden environments.

Realization of results

One of the main obstacles to bringing the presented approach into practice is the typically long process of safety certification. Although many industries have established formal methods to a certain degree—see, e.g., avionics (*87*), railway systems (*88*), and ground vehicles (*89*)—formal methods for robotic manufacturing systems are still in their infancy. However, this makes our proposed approach especially appealing to certification agencies: Instead of certifying each robot anew, one only has to certify the software for self-verification once to ensure safe operation with respect to human-robot contact. The software for self-verification could be verified using theorem proving, as it has already been done for reachability analysis (*90*).

Further applications

Our presented approach can be used in principle for many other applications. We could extend our approach to robots with parallel kinematics. Our approach also works for compliant joints by replacing the controller generation with those for compliant joints (*91*-*93*), which could be particularly useful in non-industrial settings. In addition, the safety concept could be used for service robots in households once human poses can be reliably tracked. Our online verification concept has also been applied to predict pedestrians for mobile robots (*94*) and automated vehicles (*95*).

MATERIALS AND METHODS

Validation of human models and their occupancy prediction

We have evaluated the models of adults used according to the requirements that the occupancy prediction of these models must be over-approximative, tight, and quick to compute. Our validation did not consider children because they are not allowed to work in factories. The degree of conservatism was tested twofold: We performed extensive conformance checking (*96*), and we explored the space of physically possible movements using a biomechanical model (*97*).

The data we used for validation were from the publicly available motion-capture database of the Carnegie Mellon University Graphics Lab. Movements include everyday movements—e.g., construction work, machining work, manipulating objects, and stumbling—as well as sports-related, dance-related, and acrobatic movements. Because sports, dance, and acrobatic movements are forbidden in a factory, we excluded them from our analysis. To try to find extreme movements not present in recorded data for the joint-space models in (*73*), we used the high-fidelity biomechanical model in (*97*) to systematically explore the state space using rapidly exploring random trees (*98*), which explore previously unexplored regions more efficiently compared with random testing (*99*).

For each movement in the database, we computed the occupancy for the duration of the recorded movement starting from its initial state. All behaviors of the dataset and from (*98*) are enclosed by our predicted occupancy. To demonstrate that our results are not overly conservative, we reduced the parameters for maximum velocity and maximum acceleration in half, which resulted in behaviors that were not enclosed.

Setup of the robot in the user study

The predefined long-term plans of the robot were three template movements, where the robot moved outward (in the y direction) for 1.7 s and inward again for 1.7 s, at three different x-axis locations, as shown in Fig. 5. The robot executed these movements in random order, moving between the start positions of the three template movements randomly and continuously until the human had finished his or her task.

Two possible modes of safe operation permitted in ISO 10218-1 (*3*) are the *safety-rated monitored stop* and *speed and separation monitoring*. The response of the robot to the human entering its workspace was one of the following:

- *Safety-rated monitored stop*: The robot stops when any part of the human enters a static safety zone.
- *Speed and separation monitoring*: The robot performs self-verification as described above, reducing or recovering its speed as a result of being verified unsafe or safe, respectively.

Static safety zone. A static safety zone is a standard approach in industry: When a human enters the workspace of the robot, it stops immediately. The workspace is enclosed by an axis-aligned bounding box enclosing the entire possible occupancy of the robot during the programmed movements. This box is extended in each direction by $v_{max,human}T + C$, taken from ISO 13855 (*2*), where $v_{\max,human}$ is the maximum velocity of the human, taken as 2 m/s as in (*2*); T is total time the robot requires to come to a stop, including the latency and communications delay in the sensing loop and the calculation time; and C is the penetration distance of the sensing technology before a human is detected (e.g., penetration due to resolution of light curtain; because we used infrared markers, C is zero).

Self-verification. The static safety zone is based on the assumption that the human moves no faster than a maximum speed $v_{\max,human}$. For a fair comparison, the self-verification is based on the first-order model $\mathcal{R}_y^{(1)}(t) = \mathbf{y}(0) \oplus B(\delta y) \oplus B(v_{y,max} \cdot t)$ as introduced previously with the maximum speed of the human also taken as 2 m/s, as stated in ISO 13855.

Statistical methods for user study

To test that the robot was significantly faster in the self-verification approach, we took the mean robot idle times for each participant over the last four trials, the first four trials, and over all trials and tested the data for normality with the Shapiro-Wilk test (*100*). The assumption of normality did not apply to the robot times of the last four trials ($p < 0.05$ that the data are normally distributed); thus, to ensure validity of the results, we used the Kruskal-Wallis H test (*101*).

Supplementary Materials

Supplementary Text
Movie S1. Interplay of self-programming and self-verification.
Movie S2. Comparison of optimal IMPROV compositions with commercial robots.
Data File S1. Robot hardware.
Data file S2. Software.
Data file S3. User study evaluation.
References (102-107)

References and Notes

1. A. Brunete *et al.*, Current trends in reconfigurable modular robots design. *International Journal of Advanced Robotic Systems* **14**, 1-21 (2017).
2. International Organization for Standardization, Safety of machinery - Positioning of safeguards with respect to the approach speeds of parts of the human body, ISO Standard 13855:2010 (2010).
3. International Organization for Standardization, Robots and robotic devices - Safety requirements for industrial robots - Part 1: Robots, ISO Standard 10218-1:2011 (2011).
4. Reported fatal and non-fatal injuries in Great Britain by kind of accident and broad industry group 2013-2018, http://www.hse.gov.uk/statistics/tables/ridkind.xlsx, accessed: 27.02.19
5. B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, *Robotics: Modelling, Planning and Control*. (Springer, 2009).
6. R. Kelly, Global positioning of robot manipulators via PD control plus a class of nonlinear integral actions. *IEEE Transactions on Automatic Control* **43**, 934-938 (1998).

7. M. Liu, Decentralized control of robot manipulators: nonlinear and adaptive approaches. *IEEE Transactions on Automatic Control* **44**, 357-363 (1999).
8. W. W. Melek, A. A. Goldenberg, Neurofuzzy control of modular and reconfigurable robots. *IEEE/ASME Transactions on Mechatronics* **8**, 381-389 (2003).
9. M. Zhu, Y. Li, Decentralized Adaptive Fuzzy Control for Reconfigurable Manipulators, Proc. IEEE Conference on Robotics, Automation and Mechatronics, 404-409 (2008).
10. Y. Tang, M. Tomizuka, G. Guerrero, G. Montemayor, Decentralized robust control of mechanical systems. *IEEE Trans. on Automatic Control* **45**, 771-776 (2000).
11. G. Liu, S. Abdul, A. A. Goldenberg, Distributed Modular and Reconfigurable Robot Control with Torque Sensing, Proc. IEEE Int. Conf. on Mechatronics and Automation, 384-389 (2006).
12. Z. Li, W. W. Melek, C. Clark, Decentralized robust control of robot manipulators with harmonic drive transmission and application to modular and reconfigurable serial arms. *Robotics* **27**, 291-302 (2009).
13. W.-H. Zhu, *Virtual decomposition control: toward hyper degrees of freedom robots*. (Springer Science & Business Media, 2010), vol. 60.
14. W.-H. Zhu *et al.*, Precision Control of Modular Robot Manipulators: The VDC Approach With Embedded FPGA. *IEEE Transactions on Robotics* **29**, 1162-1179 (2013).
15. C. J. J. Paredis, H. B. Brown, P. K. Khosla, A rapidly deployable manipulator system, Proc. IEEE Int. Conf. on Robotics and Automation, 1434-1439 (1996).
16. I.-M. Chen, G. Yang, Automatic Model Generation for Modular Reconfigurable Robot Dynamics. *Journal of Dyn. Sys., Meas., Control* **120**, 346-352 (1998).
17. I.-M. Chen, S. H. Yeo, G. Chen, G. Yang, Kernel for Modular Robot Applications: Automatic Modeling Techniques. *Int. J. Robot. Res.* **18**, 225-242 (1999).
18. E. Meister, E. Nosov, P. Levi, Automatic onboard and online modelling of modular and self-reconfigurable robots, Proc. IEEE Conference on Robotics, Automation and Mechatronics, 91-96 (2013).
19. F. C. Park, J. E. Bobrow, S. R. Ploen, A Lie Group Formulation of Robot Dynamics. *Int. J. Robot. Res.* **14**, 609-618 (1995).
20. Z. M. Bi, W. J. Zhang, I. M. Chen, S. Y. T. Lang, Automated generation of the D-H parameters for configuration design of modular manipulators. *Robotics and Computer-Integrated Manufacturing* **23**, 553-562 (2007).
21. J. Denavit, R. S. Hartenberg, A kinematic notation of lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics* **22**, 215-221 (1955).
22. L. Kelmar, P. K. Khosla, Automatic generation of kinematics for a reconfigurable modular manipulator system, Proc. IEEE Int. Conf. on Robotics and Automation, 663-668 (1988).
23. B. Benhabib, G. Zak, M. G. Lipton, A Generalized Kinematic Modeling Method for Modular Robots. *Journal of Robotic Systems* **6**, 545-571 (1989).
24. S. Haddadin, A. Albu-Schäffer, G. Hirzinger, Safety Evaluation of Physical Human-Robot Interaction via Crash-Testing, Proc. Robotics: Science and Systems, 217-224 (2007).
25. S. Haddadin, A. Albu-Schäffer, A. De Luca, G. Hirzinger, Collision detection and reaction: A contribution to safe physical Human-Robot Interaction, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 3356-3363 (2008).
26. L. Villani, J. De Schutter, in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib, Eds. (Springer, 2016), chap. Force Control, pp. 195-220.
27. R. Rossi, M. P. Polverini, A. M. Zanchettin, P. Rocco, A pre-collision control strategy for human-robot interaction based on dissipated energy in potential inelastic impacts, Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 26-31 (2015).

28. A. Raatz, S. Blankemeyer, G. Runge, C. Bruns, G. Borchert, in *Soft Robotics: Transferring Theory to Application,* A. Verl, A. Albu-Schäffer, O. Brock, A. Raatz, Eds. (Springer, 2015), chap. Opportunities and Challenges for the Design of Inherently Safe Robots, pp. 173-183.
29. M. Grebenstein *et al.*, The DLR hand arm system, Proc. IEEE Int. Conf. Robotics and Automation, 3175-3182 (2011).
30. C. Majidi, Soft robotics: a perspective—current trends and prospects for the future. *Soft Robotics* **1**, 5-11 (2014).
31. S. Kim, C. Laschi, B. Trimmer, Soft robotics: a bioinspired evolution in robotics. *Trends in Biotechnology* **31**, 287-294 (2013).
32. J. Burgner-Kahrs, D. C. Rucker, H. Choset, Continuum Robots for Medical Applications: A Survey. *IEEE Transactions on Robotics* **31**, 1261-1280 (2015).
33. M. Althoff, Dissertation, Technische Universität München, (2010).
34. M. Althoff, B. H. Krogh, Reachability Analysis of Nonlinear Differential-Algebraic Systems. *IEEE Transactions on Automatic Control* **59**, 371-383 (2014).
35. E. Asarin *et al.*, Recent progress in continuous and hybrid reachability analysis, Proc. of the IEEE Conference on Computer Aided Control Systems Design, 1582-1587 (2006).
36. G. Frehse *et al.*, SpaceEx: Scalable Verification of Hybrid Systems, Proc. of the 23rd International Conference on Computer Aided Verification, 379-395 (2011).
37. T. Fraichard, H. Asama, Inevitable collision states—A step towards safer robots? *Advanced Robotics* **18**, 1001-1024 (2004).
38. S. Bouraine, T. Fraichard, H. Salhi, Provably safe navigation for mobile robots with limited field-of-views in dynamic environments. *Autonomous Robots* **32**, 267-283 (2012).
39. O. Khatib, in *Autonomous Robot Vehicles*, I. J. Cox, G. T. Wilfong, Eds. (Springer, 1990), chap. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, pp. 396-404.
40. F. Flacco, T. Kröger , A. De Luca, O. Khatib, A depth space approach to human-robot collision avoidance, Proc. IEEE Int. Conf. Robotics and Automation, 338-345 (2012).
41. D. H. Park, H. Hoffmann, P. Pastor, S. Schaal, Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields, Proc. IEEE-RAS Int. Conf. Humanoid Robots, 91-98 (2008).
42. M. P. Polverini, A. M. Zanchettin, P. Rocco, Real-time collision avoidance in human-robot interaction based on kinetostatic safety field, Proc. IEEE/RSJ Int. Conf. Intell. Robots and Systems, 4136-4141 (2014).
43. N. M. Ceriani, A. M. Zanchettin, P. Rocco, Collision Avoidance with Task Constraints and Kinematic Limitations for Dual Arm Robots, Proc. 13th Int. Conf Intelligent Autonomous Systems, 1285-1299 (2016).
44. J. Liu, X. Zhang, G. Hao, Survey on research and development of reconfigurable modular robots. *Advances in Mechanical Engineering* **8**, 1-21 (2016).
45. C. Wright *et al.*, Design of a modular snake robot, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2609-2614 (2007).
46. I.-M. Chen, J. W. Burdick, Enumerating the non-isomorphic assembly configurations of modular robotic systems. *The International Journal of Robotics Research* **17**, 702-719 (1998).
47. G. Yang, I.-M. Chen, Task-based optimization of modular robot configurations: minimized degree-of-freedom approach. *Mechanism and machine theory* **35**, 517-540 (2000).
48. C. J. J. Paredis, P. Khosla, Synthesis methodology for task based reconfiguration of modular manipulator systems, Proc. of the 6th International Symposium on Robotics Research, ISRR, (1993).

49. S. Farritor, S. Dubowsky, N. Rutman, J. Cole, A systems-level modular design approach to field robotics, International Conference on Robotics and Automation, 2890-2895 (1996).
50. O. Chocron, P. Bidaud, Evolutionary Algorithms in Kinematic Design of Robotic Systems, Proc. of the IEEE/RSJ International Conference on Intelligent Robot and Systems, 1111-1117 (1997).
51. J. Han, W. K. Chungt, Y. Youm, S. H. Kim, Task Based Design of Modular Robot Manipulator using Efficient Genetic Algorithm, Proc. of the IEEE International Conference on Robotics and Automation, 507-512 (1997).
52. I.-M. Chen, J. W. Burdick, Determining Task Optimal Modular Robot Assembly Configurations, Proc. of the IEEE International Conference on Robotics and Automation, 132-137 (1995).
53. A. Brunete, M. Hernando, E. Gambao, Offline GA-Based Optimization for Heterogeneous Modular Multiconfigurable Chained Microrobots. *IEEE/ASME Transactions on Mechatronics* **18**, 578-585 (2013).
54. A. Giusti, M. Althoff, On-the-Fly Control Design of Modular Robot Manipulators. *IEEE Transactions on Control Systems Technology* **26**, 1484-1491 (2018).
55. J. Y. S. Luh, M. W. Walker, R. P. C. Paul, On-line computational scheme for mechanical manipulators. *ASME Journal of Dynamic Systems, Measurement and Control* **102**, 468-474 (1980).
56. L. Sciavicco, B. Siciliano, L. Villani, Lagrange and Newton-Euler dynamic modelling of a gear-driven rigid robot manipulator with inclusion of motor inertia effects. *Advanced Robotics* **10**, 317-334 (1995).
57. A. De Luca, L. Ferrajoli, A modified Newton-Euler method for dynamic computations in robot fault detection and control, Proc. IEEE Int. Conf. on Robotics and Automation, 3359-3364 (2009).
58. M. W. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*. (Wiley, 2006).
59. J.-J. E. Slotine, W. Li, On the Adaptive Control of Robot Manipulators. *Int. J. Robot. Res.* **6**, 49-59 (1987).
60. J. S. Yuan, Closed-Loop Manipulator Control Using Quaternion Feedback. *IEEE Journal on Robotics and Automation* **4**, 434-440 (1988).
61. F. Caccavale, S. Chiaverini, B. Siciliano, Second-Order Kinematic Control of Robot Manipulators with Jacobian Damped Least-Squares Inverse: Theory and Experiments. *IEEE/ASME Transactions on Mechatronics* **2**, 188-194 (1997).
62. P. Hsu, J. Hauser, S. Sastry, Dynamic Control of Redundant Manipulators, American Control Conference, 2135-2139 (1988).
63. A. De Luca, G. Oriolo, B. Siciliano, Robot Redundancy Resolution at the Acceleration Level. *Laboratory Robotics and Automation* **4**, 97-106 (1992).
64. H. M. Choset *et al.*, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. (MIT Press, 2005).
65. S. M. LaValle, *Planning Algorithms*. (Cambridge University Press, Cambridge, U.K., 2006).
66. L. E. Kavraki, P. Svestka, J.-C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* **12**, 566-580 (1996).
67. A. Shkolnik, R. Tedrake, Path planning in 1000+ dimensions using a task-space Voronoi bias, IEEE International Conference on Robotics and Automation, 2061-2067 (2009).

68. J. Kinugawa, A. Kanazawa, S. Arai, K. Kosuge, Adaptive Task Scheduling for an Assembly Task Coworker Robot Based on Incremental Learning of Human's Motion Patterns. *IEEE Robotics and Automation Letters* **2**, 856-863 (2017).
69. J. Mainprice, R. Hayne, D. Berenson, Goal Set Inverse Optimal Control and Iterative Replanning for Predicting Human Reaching Motions in Shared Workspaces. *IEEE Transactions on Robotics* **32**, 897-908 (2016).
70. H. Liu, L. Wang, Human Motion Prediction for Human-Robot Collaboration. *Journal of Manufacturing Systems* **44**, 287-294 (2017).
71. T. Sanders, K. E. Oleson, D. R. Billings , J. Y. C. Chen, P. A. Hancock, A Model of Human-Robot Trust: Theoretical Model Development. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* **55**, 1432-1436 (2011).
72. T. Kröger, F. M. Wahl, Online Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events. *IEEE Transactions on Robotics* **26**, 94-111 (2010).
73. A. Pereira, M. Althoff, Overapproximative Human Arm Occupancy Prediction for Collision Avoidance. *IEEE Transactions on Automation Science and Engineering* **15**, 818-831 (2018).
74. R. Alur *et al.*, The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science* **138**, 3-34 (1995).
75. H. Täubig, B. Bäuml, U. Frese, Real-time swept volume and distance computation for self collision detection, Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 1585-1592 (2011).
76. C. Ericson, *Real-Time Collision Detection*. (CRC Press, 2005).
77. M. Salem, K. Dautenhahn, Evaluating Trust and Safety in HRI : Practical Issues and Ethical Challenges, Proc. of Workshop "Emerging Policy and Ethics of Human-Robot Interaction" at ACM/IEEE Int. Conf. Human Robot Interaction, (2015).
78. J. Lee, N. Moray, Trust, control strategies and allocation of function in human-machine systems. *Ergonomics* **35**, 1243-1270 (1992).
79. J. Pan, S. Chitta, D. Manocha, FCL: A general purpose library for collision and proximity queries, IEEE International Conference on Robotics and Automation, 3859-3866 (2012).
80. B. Akan, A. Ameri, B. Çürüklü, L. Asplund, Intuitive industrial robot programming through incremental multimodal language and augmented reality, Proc.of the IEEE International Conference on Robotics and Automation, 3934-3939 (2011).
81. G. F. Rossano, C. Martinez, M. Hedelind, S. Murphy, T. A. Fuhlbrigge, Easy robot programming concepts: An industrial perspective, Proc. IEEE Int. Conf. on Automation Science and Engineering, 1119-1126 (2013).
82. P. Regan, S. Hamilton, NASA's mission reliable. **37**, 59-68 (2004).
83. C. G. Atkeson *et al.*, No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge, Proc. of the 15th IEEE-RAS International Conference on Humanoid Robots, 623-630 (2015).
84. J. Inoue, F. Kanehiro, M. Morisawa, A. Mori, Detecting Errors in a Humanoid Robot, Proc. of the IEEE International Conference on Software Quality, Reliability and Security, 163-170 (2018).
85. Y. Koren *et al.*, Reconfigurable Manufacturing Systems. **48**, 527-540 (1999).
86. Z. M. Bi, S. Y. T. Lang, W. Shen, L. Wang, Reconfigurable manufacturing systems: the state of the art. *International Journal of Production Research* **46**, 967-992 (2008).
87. Y. Moy, E. Ledinot, H. Delseny, V. Wiels, B. Monate, Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. 50-57 (2013).

88. A. Chiappini *et al.*, Formalization and validation of a subset of the European Train Control System, Proc. of the 32nd ACM/IEEE International Conference on Software Engineering, 109-118 (2010).
89. H. Täubig *et al.*, Guaranteeing functional safety: design for provability and computer-aided verification. *Autonomous Robots* **32**, 303-331 (2012).
90. F. Immler, Verified Reachability Analysis of Continuous Systems, Tools and Algorithms for the Construction and Analysis of Systems, 37-51 (2015).
91. A. De Luca, Feedforward/Feedback Laws for the Control of Flexible Robots, IEEE International Conference on Robotics and Automation, 233-240 (2000).
92. A. Albu-Schäffer, C. Ott, G. Hirzinger, A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots *International Journal of Robotics Research* **26**, 23-39 (2007).
93. A. Giusti, J. Malzahn, N. G. Tsagarakis, M. Althoff, On the Combined Inverse-Dynamics/Passivity-Based Control of Elastic-Joint Robots. *IEEE Transactions on Robotics* **34**, 1461-1471 (2018).
94. S. B. Liu *et al.*, Provably Safe Motion of Mobile Robots in Human Environments, Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 1351-1357 (2017).
95. M. Koschi, C. Pek, M. Althoff, Set-Based Prediction of Pedestrians in Urban Environments Considering Formalized Traffic Rules, Proc. of the 21st IEEE Int. Conf. on Intelligent Transportation Systems, 2704-2711 (2018).
96. H. Roehm, J. Oehlerking, M. Woehrle, M. Althoff, Reachset Conformance Testing of Hybrid Automata, Hybrid Systems: Computation and Control, 277-286 (2016).
97. S. L. Delp *et al.*, OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement. *IEEE Transactions on Biomedical Engineering* **54**, 1940-1950 (2007).
98. C. Stark, A. Pereira, M. Althoff, Reachset Conformance Testing of Human Arms with a Biomechanical Model, IEEE International Conference on Robotic Computing, 209-216 (2018).
99. T. Dang, T. Nahhal, Coverage-guided Test Generation for Continuous and Hybrid Systems. *Formal Methods in System Design* **34**, 183-213 (2009).
100. S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples). *Biometrika* **52**, 591-611 (1965).
101. W. H. Kruskal, W. A. Wallis, Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association* **47**, 583-621 (1952).
102. F. Caccavale, C. Natale, B. Siciliano, L. Villani, Resolved-acceleration Control of Robot Manipulators: A Critical Review with Experiments. *Robotica* **16**, 565-573 (1998).
103. J. S. Yuan, Closed-loop manipulator control using quaternion feedback. *IEEE Journal on Robotics and Automation* **4**, 434-440 (1988).
104. M. Althoff, An introduction to CORA 2015, 1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems, 120-151 (2015).
105. J. Jin, N. Gans, Parameter identification for industrial robots with a fast and robust trajectory design approach. *Robotics and Computer-Integrated Manufacturing* **31**, 21-29 (2015).
106. A. Jubien, M. Gautier, A. Janot, Dynamic identification of the Kuka LWR robot using motor torques and joint torque sensors data. *IFAC Proceedings Volumes* **47**, 8391-8396 (2014).
107. C. Gaz, F. Flacco, A. De Luca, Extracting feasible robot parameters from dynamic coefficients using nonlinear optimization methods, IEEE Int. Conf. on Robotics and Automation, 2075-2081 (2016).

Acknowledgments

We thank the team of graduate and undergraduate students who worked at the Cyber-Physical Systems Group at the Technical University of Munich for constructing the IMPROV modules. In particular, we thank M. Baumann and J. Gerstner for contributing to the design of the user study and carrying it out. We also thank D. Beckert for the software development.

Funding

The work was supported by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Program FP7/2007-2013/ under REA grant 608022 and by the Central Innovation Programme of the German Federal Government under grant ZF4086004LP7.

Author contributions

M.A. led the research project and is the main author. A.G. developed the self-programming capabilities. S.B.L. realized the selection of optimal compositions. A.P. developed the self-verification capabilities.

Competing interests

S.B.L. and M.A. have filed a European patent under application number EP18201371.4 for collision avoidance of reconfigurable modular robots. The other authors declare that they have no competing interests.

Data and materials availability

All data needed to evaluate the conclusions in the paper are present in the paper or the Supplementary Materials. Other data for this study have been deposited in https://bitbucket.org/MatthiasAlthoff/improv/src/master.

Science Robotics
AAAS

robotics.sciencemag.org/cgi/content/full/4/31/eaaw1924/DC1

Supplementary Materials for

Effortless creation of safe robots from modules through self-programming and self-verification

M. Althoff*, A. Giusti, S. B. Liu, A. Pereira

*Corresponding author. Email: althoff@tum.de

Published 19 June 2019, *Sci. Robot.* **4**, eaaw1924 (2019)
DOI: 10.1126/scirobotics.aaw1924

The PDF file includes:

Other Supplementary Material for this manuscript includes the following:

(available at robotics.sciencemag.org/cgi/content/full/4/31/eaaw1924/DC1)

Movie S1 (.mp4 format). Interplay of self-programming and self-verification.
Movie S2 (.mp4 format). Comparison of optimal IMPROV compositions with commercial robots.
Data file S1 (.zip format). Robot hardware.
Data file S2 (.zip format). Software.
Data file S3 (.zip format). User study evaluation.

Text

In this supplementary text, we present in more detail the hardware used, the self-programming of the robot, and the experimental data of the user study.

Hardware

The basis of IMPROV are modules of the Schunk Powerball Lightweight Arm LWA 4P. These modules have been modified such that none of the original position control schemes are used anymore and we only use the mechanical parts of the robot of which we know the parameters. In this subsection, we describe the robot, how we designed new link modules, and how we sensed the human pose. The overall setup is shown in Fig. S1 and in the schematic in Fig. S2.

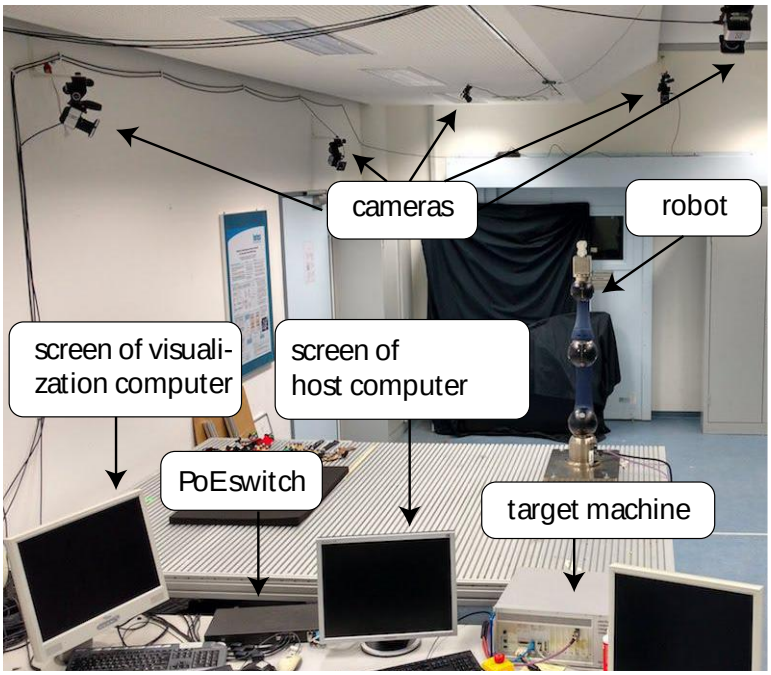

Fig. S1. Experimental setup.

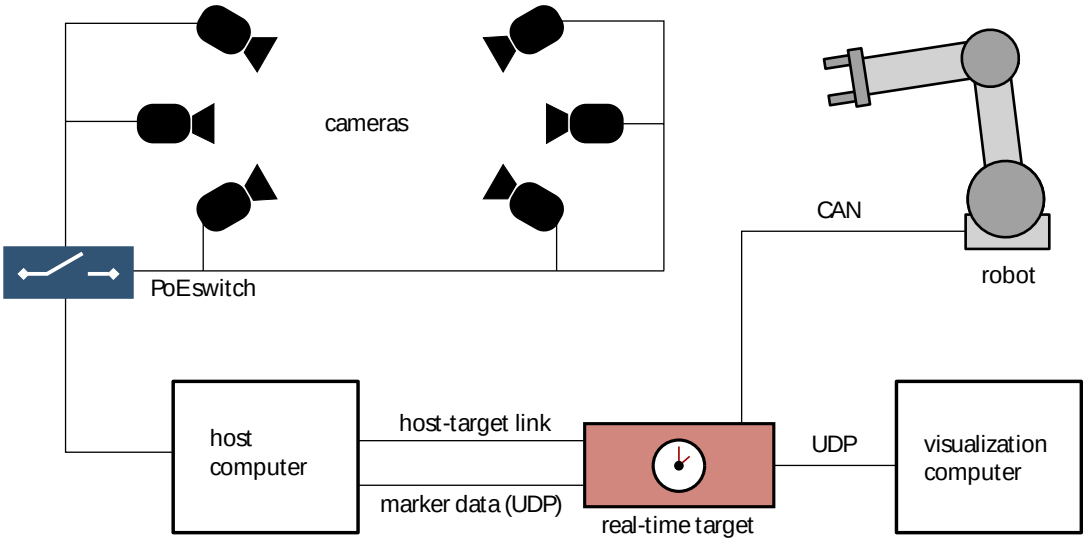

Fig. S2. Schematic diagram of the experimental setup.

Schunk Powerball Lightweight Arm LWA 4P

The robot used in our experiments is a heavily modified Schunk Powerball Lightweight Arm LWA 4P. The robot has six degrees of freedom, and we attached the PG70 gripper from Schunk to it. Although the robot has modular mechanics, it cannot be used as a modular robot since changing the configuration requires manually reprogramming the robot, which is not possible since the software is closed source. How we modified the robot so that it can program itself is described in the section *Software for Self Programming*. Further data is summarized in Table S1.

Table S1. Technical data of Schunk Powerball Lightweight Arm LWA 4P.

Max. payload load [kg]	6	
Repeat accuracy [mm]	±0.15	
Position feedback	Pseudo-absolute position measuring	
Drives	Brushless servomotors with permanent-magnet brake	
Pan-tilt unit flange	Flat tool changer with free lines and power supply	
Installation direction	Any	
Dead weight [kg]	15	
IP class [IP]	40	
Power supply	24 V DC / avg. 3 A / max. 12 A	
Interface	CANopen (CiA DS402:IEC61800-7-201)	
Axes	**Max. speed with nominal load [deg/s]**	**Range [deg]**
Axis 1	72	±170
Axis 2	72	±170
Axis 3	72	±155.5
Axis 4	72	±170
Axis 5	72	±170
Axis 6	72	±170

CAD Files of Newly Developed Links

For IMPROV, we have used existing modules from Schunk and modules that we have produced ourselves. A selection of modules used and a possible assembly is shown in Fig. S3.

Fig. S3. Selection of modules (left) and a possible assembly (right).

The link modules that we produced ourselves fulfill the following specifications:

- Mechanical robustness: For arbitrary assemblies, the parts do not fail mechanically under the assumption that the assembled robot is movable given the torque limits of the motors.
- Stiffness: The stiffness of the links is comparable to the ones from Schunk to avoid too large a positioning error at the end effector.
- Full functionality: The extension has to allow for complete signal transmission and power supply.
- Protection of electrical connections: The extensions provide physical protection to the signal and power cables as well as to any other connecting elements, such as PCBs.

An example of a mechanical test for one of our developed links is shown in Fig. S4.

Fig. S4. Link module and its finite element analysis to test mechanical strength.

Central Control Unit

The central control unit that implements the presented framework is a Speedgoat Real-Time target PC, equipped with an Intel Core i7-3770K clocked at 3.5 GHz and 4 GB of RAM. The communication with the robot is done via CAN-bus. The communication bus limits the sampling rate that can be used for centralized control to 500Hz when assembling a six-axes robot. The controller has been developed using MATLAB/Simulink 2015b and has been implemented on

the target machine via automatic code generation with MATLAB Coder, Simulink Coder, and Simulink Real-Time toolboxes.

Tracking System

The human is tracked by a six-camera Vicon Vero 1.3 motion capture system, operating at 250Hz. The cameras operate by shining infrared light on the workspace, which is reflected by clusters of retroreflective markers on the objects to be tracked. The six cameras are connected via a Power over Ethernet (PoE) switch to a Dell Precision Tower 3620 with an Intel Xeon E3-1270v5 processor (3.6GHz), 16GB RAM, and NVIDIA Quadro K620 Graphics card running Windows 10. The camera data is processed by Vicon Tracker 3.5 software on this machine to triangulate the positions of the marker clusters; these are then sent via User Datagram Protocol (UDP), at 1kHz, to the real-time target machine. The marker clusters on the human body are shown in Fig. S5.

In the experiments with the replanning of trajectories, we also have a visualization in a virtual environment created in Coppelia V-REP and receive information in real time from the target machine via UDP at a rate of 20Hz.

The processing of the data in Tracker 3.5 is the only non-real-time part of the system; to the best of the authors' knowledge, no tracking system with deterministic latencies near comparable to the latencies of the Vicon system exist. According to the information from the developers[1], the Tracker 3.5 software has a computation time of 1.5ms for five objects and 2.8ms for ten objects. As we had seven or eight objects on the human (depending on whether the marker on the lower back was used), we took the latency at a conservative 5ms, plus the frame rate of the cameras and the cycle time of the UDP connection, which were 4ms and 1ms respectively, i.e., 10ms overall.

Fig. S5. Marker cluster positions on a human participant.

Software for Self-Programming

Deactivation of Internal Control Circuits

[1] www.vicon.com/products/software/tracker, accessed 2.1.2018

The joint modules used are the commercially available Schunk Powerballs. These actuation modules are natively controlled with a cascaded proportional-integral-derivative scheme. Unfortunately, details of the native schemes are not publicly available.

In order to be able to deploy our automatically generated model-based controllers, without any interference from the built-in controllers, all built-in controllers were deactivated. This was achieved by setting the gain associated with the velocity error feedback to zero. After disabling the built-in decentralized controllers, we enabled a feedforward term from the actuators control unit, which allows one to directly set current commands to the driver of the motors.

Implementation of Self-Programming

Automatic deployment of both joint space and task space controllers is performed automatically using collected module data (**ModRob**) using the architecture in Fig. S6. Assembled robot descriptions, both for kinematics (**DH**) and dynamics (**DynPar**), are generated automatically using the two MATLAB functions "ModRob2DH" and "ModRob2DynPar", which are provided in the data file S2.
The central control unit uses the obtained models to self-program model-based control methods. The MATLAB code used for self-programming after each new assembly has been included in the data file S2.

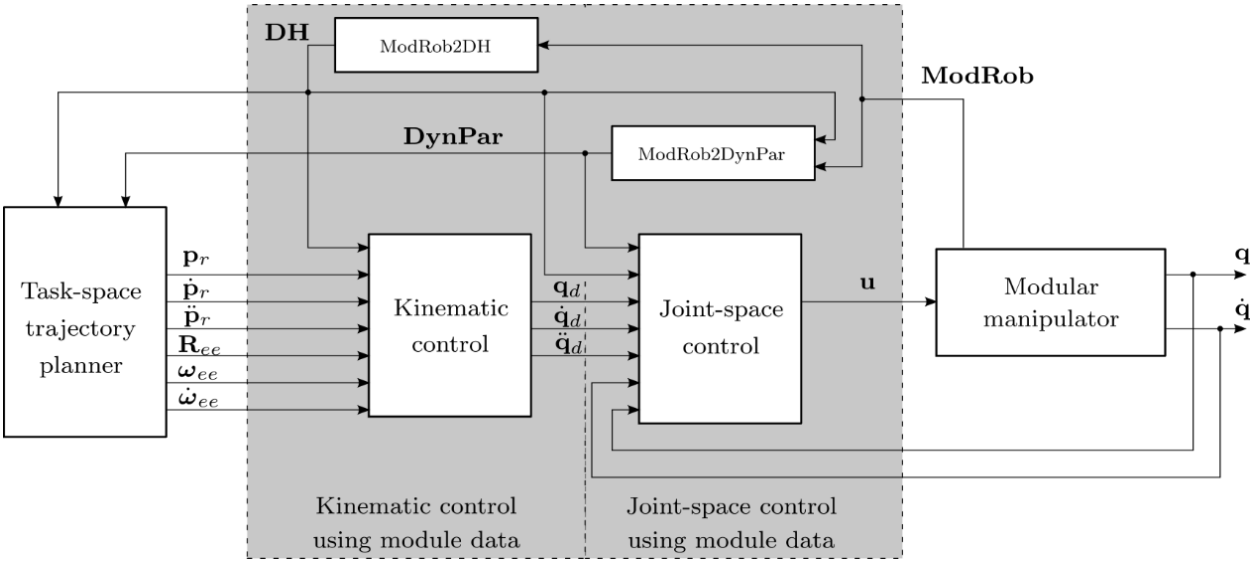

Fig. S6. Modular robot control architecture with self-programming capabilities.

In addition to the blocks of kinematic and joint-space control as in a classical structure (see e.g., (*5*)), the blocks for processing the module data are added to deliver the assembled-robot data. This combination enables the automatic generation of the control of the arm, once the robot is assembled and **ModRob** is created. This architecture is simple: it allows tracking of task space trajectories by solving the inverse kinematic problem online (using closed-loop inverse-kinematics schemes) to obtain the reference trajectories in joint space ($\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\ddot{\mathbf{q}}_d$) that are tracked by means of a joint-space tracking controller. Both the kinematic control and joint-space

control are automatically deployed using the automatically generated robot description. The following subsections detail how the kinematic and joint-space control are implemented.

Joint-Space Control

We have implemented self-verification for two of the most effective model-based joint space controllers: inverse dynamics control (see e.g., Sec. 8.5.2 of (*5*)) and passivity-based control (see e.g., Sec. 8.4 of (*58*)). To properly handle assembly-dependent friction model uncertainties, we have also implemented a version of the passivity-based tracking controller with adaptive friction compensation. We demonstrate the effectiveness of these self-programmed controllers below.

Inverse dynamics control

The inverse dynamics control exploits the model knowledge to directly cancel couplings through feedback and obtain a linear and decoupled system from a new auxiliary control input variable $\mathbf{y}$. The inverse dynamics control law is implemented using

$$\mathbf{u}_{ID} = \mathbf{M}(\mathbf{q})\mathbf{y} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}),$$

which provides $\ddot{\mathbf{q}} = \mathbf{y}$. A typical choice of $\mathbf{y}$ is

$$\mathbf{y} = \ddot{\mathbf{q}}_d + \mathbf{K}_D\dot{\mathbf{e}} + \mathbf{K}_P\mathbf{e},$$

providing

$$\ddot{\mathbf{e}} + \mathbf{K}_D\dot{\mathbf{e}} + \mathbf{K}_P\mathbf{e} = \mathbf{0},$$

where $\mathbf{K}_P$, $\mathbf{K}_D$ are diagonal positive definite gain matrices of proper dimensions. By using our automatically generated robot description, the inverse dynamics controller is efficiently implemented as follows:

$$\mathbf{u}_{ID} = \mathbf{M}(\mathbf{q})\mathbf{y} + \mathbf{c}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = NEA_g\left(\mathbf{q},\dot{\mathbf{q}},\mathbf{y},\mathbf{DH},\mathbf{DynPar}\right).$$

The tuning of the above controller is rather simple. The control parameters $\mathbf{K}_P = \omega_n^2\mathbf{I}$ and $\mathbf{K}_D = 2\zeta\omega_n\mathbf{I}$ are set by a user-defined natural frequency ω_n and damping ratio ζ.

Passivity-based control

Contrary to the inverse dynamics control scheme, passivity-based tracking controllers do not rely on the complete cancellation of all couplings through feedback so that this approach is typically more robust to model uncertainties compared to inverse dynamics control. Our passivity-based tracking controller is implemented using

$$\mathbf{u}_{PB} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_a + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_a + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \Lambda\mathbf{r}, \quad (1)$$

with $\dot{\mathbf{q}}_a = \dot{\mathbf{q}}_d + \mathbf{K}_r\mathbf{e}$, $\mathbf{r} = \dot{\mathbf{e}} + \mathbf{K}_r\mathbf{e}$, and where Λ and $\mathbf{K}_r$ are diagonal positive definite matrices of proper dimensions. By applying the passivity-based control law in (1) to the system

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{u}, \quad (2)$$

the following closed-loop system is obtained:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r} + \mathbf{\Lambda r} = \mathbf{0}.$$

The derivation of global stability is presented in (*58*). As for the inverse-dynamics controller, the passivity-based controller is implemented using the recursive Newton-Euler algorithm as presented in the paper. However, $NEA_g(\mathrm{L})$ has to be slightly modified according to (*57*) since it cannot compute $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_a$. The modified version is denoted by $NEA_g^*(\mathrm{L})$ and together with **DH** and **DynPar**, our approach automatically computes the following passivity-based controller:

$$\mathbf{u}_{PB} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_a + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_a + \mathbf{f}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \mathbf{\Lambda r} = NEA_g^*\left(\mathbf{q},\dot{\mathbf{q}},\dot{\mathbf{q}}_a,\ddot{\mathbf{q}}_a,\mathbf{DH},\mathbf{DynPar}\right) + \mathbf{\Lambda r}.$$

Passivity-based control with adaptive friction compensation

The load at the joints affects the friction. We address this aspect by introducing an adaptive term to the passivity-based feedback control law. Let us introduce the friction model

$$\mathbf{f}(\dot{\mathbf{q}}) = \boldsymbol{\beta}_v\dot{\mathbf{q}} + \boldsymbol{\beta}_c\mathrm{sign}(\dot{\mathbf{q}}),$$

with viscous $\boldsymbol{\beta}_v$ and static $\boldsymbol{\beta}_c$ friction coefficients, and assuming that only nominal parameters $\beta_{0v,i}$ and $\beta_{0c,i}$ for the i$^{\text{th}}$ joint are available for control, the passivity-based control command with adaptive friction compensation is computed as

$$\mathbf{u}_{PBAFC} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_a + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_a + \hat{\mathbf{f}}(\dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) + \mathbf{\Lambda r}, \quad (3)$$

where

$$\hat{f}_i(\dot{q}_i) = \hat{\beta}_{v,i}(t)\dot{q}_i + \hat{\beta}_{c,i}(t)\mathrm{sign}(\dot{q}_i), \quad \forall i \in \{1,\ldots,N\}$$

and $\hat{\beta}_{v,i}(0) = \beta_{0v,i}$, $\hat{\beta}_{c,i}(0) = \beta_{0c,i}$. Using (3) in (2), it can be shown that

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{r}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\mathbf{r} + \mathbf{\Lambda r} = \mathbf{f}(\dot{\mathbf{q}}) - \hat{\mathbf{f}}(\dot{\mathbf{q}}) = \mathbf{Y}(\dot{\mathbf{q}})\boldsymbol{\Delta}_\beta,$$

where

$$\mathbf{Y}(\dot{\mathbf{q}}) = \begin{bmatrix} \dot{q}_1 & \mathrm{sign}(\dot{q}_1) & \ldots & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & \ldots & \dot{q}_N & \mathrm{sign}(\dot{q}_N) \end{bmatrix}, \quad \boldsymbol{\Delta}_\beta = \boldsymbol{\beta} - \hat{\boldsymbol{\beta}}(t) = \begin{bmatrix} \beta_{v,1} - \hat{\beta}_{v,i}(t) \\ \beta_{c,1} - \hat{\beta}_{c,i}(t) \\ \mathrm{M} \\ \beta_{v,N} - \hat{\beta}_{v,N}(t) \\ \beta_{c,N} - \hat{\beta}_{c,N}(t) \end{bmatrix}.$$

Global asymptotic stability follows from a similar argument of Sec. 8.5.4 in (*5*), where the main difference is in the choice of $\mathbf{Y}(\dot{\mathbf{q}})$, provided that the parameters $\hat{\beta}_{v,i}(t)$ and $\hat{\beta}_{c,i}(t)$ are computed using the following adaptive law:

$$\dot{\boldsymbol{\Delta}}_\beta = -\dot{\hat{\boldsymbol{\beta}}}(t) = -\mathbf{K}_{\Delta_\beta}^{-1}\mathbf{Y}(\dot{\mathbf{q}})^T\mathbf{r},$$

with $\mathbf{K}_{\Delta_\beta}$ being a positive definite matrix of proper dimensions.

Kinematic Control

The only missing component for generating a complete controller is the consideration of the solution of the inverse kinematic problem. An analytical solution of the inverse kinematic problem can be found only for manipulators that are sufficiently simple and have specific geometries (*5*), so that we use numerical approaches. One of these approaches is based on using unit quaternions, which have unique advantages compared to Euler angles and axis-angle representations. This approach only requires the computation of the $6\times N$ geometric Jacobian $\mathbf{J}(\mathbf{q}) = [\mathbf{J}_p(\mathbf{q})^T, \mathbf{J}_\omega(\mathbf{q})^T]^T$ and its derivative $\dot{\mathbf{J}}(\mathbf{q},\dot{\mathbf{q}})$, which can be computed online by using $\mathbf{q}$, $\dot{\mathbf{q}}$ and the automatically obtained kinematic description of the robot. An implementation of the algorithm can be found in the repository in the data file S2 under "kin_fcns/dJacobian".

Closed-loop inverse kinematics schemes are usually separated into first- and second-order ones. First-order schemes provide the joint velocities and positions which let the end effector track a task-space trajectory. Drift is counteracted by employing a feedback control loop on position. However, for the purpose of this framework, the computation of the reference joint acceleration is required. In the case that a first-order scheme is employed, a practical way for computing the joint acceleration is via numerical differentiation. In contrast, second-order methods allow direct computation of the joint reference accelerations. Joint velocities and joint positions are then obtained through integration. In this case, drift is counteracted by employing a feedback control loop on both velocity and position.

Hereafter, a unit quaternion is denoted by a vector of four components, e.g., $\mathbf{Q} = [\eta, \boldsymbol{\epsilon}^T]^T = [\eta, \epsilon_x, \epsilon_y, \epsilon_z]^T$, where η is the scalar part of the quaternion and $\boldsymbol{\epsilon} = [\epsilon_x, \epsilon_y, \epsilon_z]^T$ its vectorial part. By considering the required trajectory of the end effector frame specified for positions $\mathbf{p}_r \in \mathbb{R}^3$, orientations $\mathbf{Q}_r = [\eta_r, \boldsymbol{\epsilon}_r]^T \in \mathbb{R}^4$, linear velocities and accelerations ($\dot{\mathbf{p}}_r$, $\ddot{\mathbf{p}}_r$), and angular velocities and accelerations ($\boldsymbol{\omega}_r$, $\dot{\boldsymbol{\omega}}_r$), the following closed loop inverse kinematic scheme can be implemented:

$$\ddot{\mathbf{q}}_d = \mathbf{J}^\dagger(\mathbf{q}_d)\left(\mathbf{v} - \dot{\mathbf{J}}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_d\right) - \kappa\left(\mathbf{I} - \mathbf{J}^\dagger(\mathbf{q}_d)\mathbf{J}(\mathbf{q}_d)\right)\dot{\mathbf{q}}_d, \quad (4)$$

with

$$\mathbf{v} = \begin{bmatrix} \ddot{\mathbf{p}}_r + K_v(\dot{\mathbf{p}}_r - \mathbf{J}_p(\mathbf{q}_d)\dot{\mathbf{q}}_d) + K_p(\mathbf{p}_r - \mathbf{p}_{fk}(\mathbf{q}_d)) \\ \dot{\boldsymbol{\omega}}_r + K_\omega(\boldsymbol{\omega}_r - \mathbf{J}_\omega(\mathbf{q}_d)\dot{\mathbf{q}}_d) + K_o\mathbf{e}_o(\mathbf{q}_d) \end{bmatrix}.$$

In (4), $\mathbf{J}^\dagger$ is the Jacobian pseudo-inverse (or damped least-squares inverse near the kinematic singularities); κ, K_v, K_p, K_ω, K_0 are positive gains; $\mathbf{p}_{fk}(\mathbf{q}_d)$ is the position of the end effector computed with the forward kinematics; and finally

$$\mathbf{e}_o = \eta_{fk}(\mathbf{q}_d)\boldsymbol{\epsilon}_r - \eta_r\boldsymbol{\epsilon}_{fk}(\mathbf{q}_d) - \mathbf{S}(\boldsymbol{\epsilon}_r)\boldsymbol{\epsilon}_{fk}(\mathbf{q}_d)$$

is the quaternion-based orientation error feedback vector, where $\eta_{fk}(\mathbf{q}_d)$ and $\epsilon_{fk}(\mathbf{q}_d)$ are the components of the unit quaternion for the orientation of the end effector computed using the forward kinematic function and the current joint variables of the inverse kinematic solution. The feedback variables $\mathbf{q}_d$ and $\dot{\mathbf{q}}_d$ are obtained by integrating $\ddot{\mathbf{q}}_d$. In the event that the assembled robot is redundant, second-order schemes suffer from floating null-space motions. To account for these cases, the last term on the right-hand side of (4) is added to introduce damping in the null space as in (*63*). For the considered scheme, asymptotic stability of both position and orientation error dynamics can be shown by Lyapunov arguments (see e.g., (*60, 102*)). It is worth stressing that in principle, no user intervention is required after reassembling the robot with this scheme, since the kinematic description of the robot can be automatically obtained and directly used for computing the geometric Jacobian. An implementation of (4) is available in the data file S2 under "kin_fcns/invkin_2d".

To consider saturation limits of actuators, we find the appropriate time-scaling factors for the task-space trajectories.

Validation by Experiments

Fig. S7 shows some example modules used in our experiments. A complete list of the module data can be found in the data file S2 under "/modulesDB/IMPROV_SchunkLWA4p".

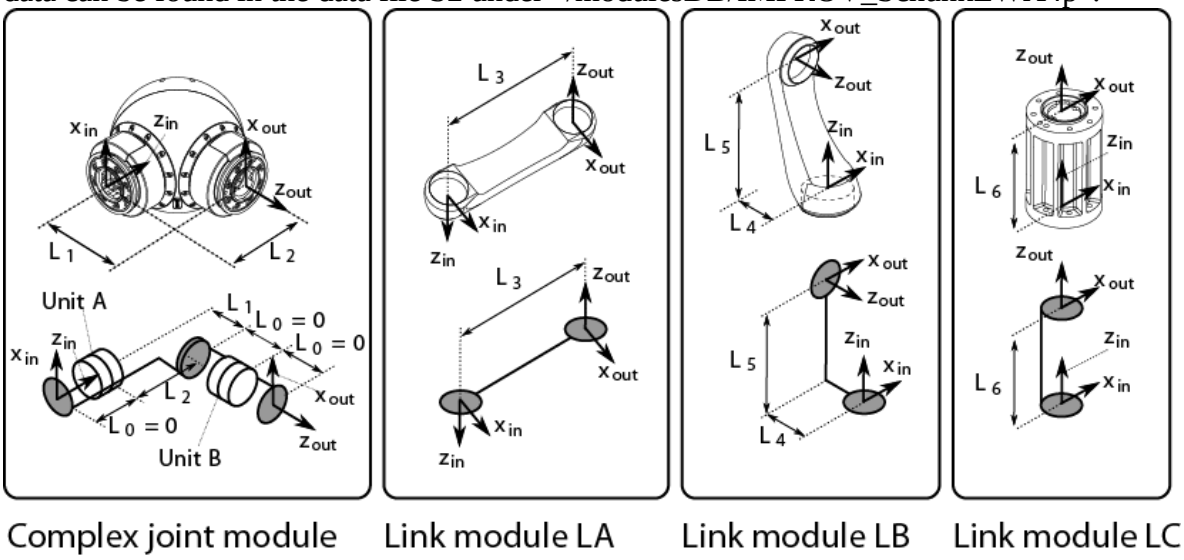

Fig. S7. Characterization of the modules using simple modular units. The models of the components have been derived using the CAD data available from the website of the robot manufacturer.

Complex joint modules, as shown on the left of Fig. S7, are decomposed into two simple joint model units (Unit A and Unit B). To parametrize these two units, the corresponding input and output frames should first be placed. The required module data has been obtained from CAD software and the data sheets from the robot manufacturer. The data relative to the actuators, such as friction parameters and rotor inertia, have been estimated by performing simple identification of each joint module using the following model:

$$I_m \sigma_r^2 \ddot{q}_i + \beta_{v,i} \dot{q}_i + \beta_{c,i} \text{sign}(\dot{q}_i) = u_i, \quad I_{eq} := I_m \sigma_r^2.$$

The parameters of the above model have been estimated by using the following regression after obtaining k samples from a test motion as follows:

$$[I_{eq}, \beta_{v,i}, \beta_{c,i}]^T = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{b},$$

where

$$\mathbf{\Phi} = \begin{pmatrix} \ddot{q}_i(1) & \dot{q}_i(1) & \text{sign}(\dot{q}_i(1)) \\ \vdots & \vdots & \vdots \\ \ddot{q}_i(k) & \dot{q}_i(k) & \text{sign}(\dot{q}_i(k)) \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} u_i(1) \\ \vdots \\ u_i(k) \end{pmatrix}.$$

The acceleration was obtained from zero-phase-shift digital filtering as implemented in the MATLAB signal processing toolbox. The results of the identification are shown in Fig. S8.

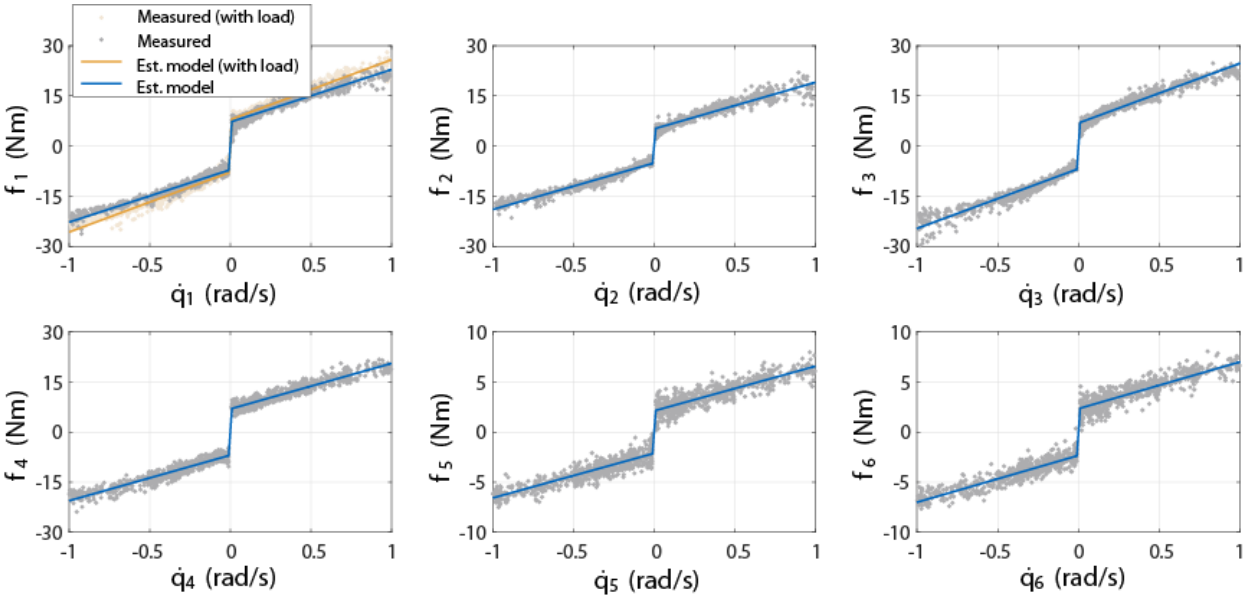

Fig. S8. Results of the friction identification procedure for all joint modules. This figure additionally shows the influence of the load for the first joint axis.

Validation of the Automatically Generated Models

To validate the automatically-obtained models, we let the robot follow a trajectory and compare the measured joint positions, velocities, and applied control torques with those of the models. We estimate the torque using the recursive N-E algorithm, which takes as input the automatically-generated kinematic/dynamic description of the assembled robot, the measured joint position, velocity, and acceleration vector. Since the joint acceleration is typically not available, we processed the measured joint position data offline through numerical differentiation and zero-phase-shift digital filtering as implemented in the MATLAB signal processing toolbox.

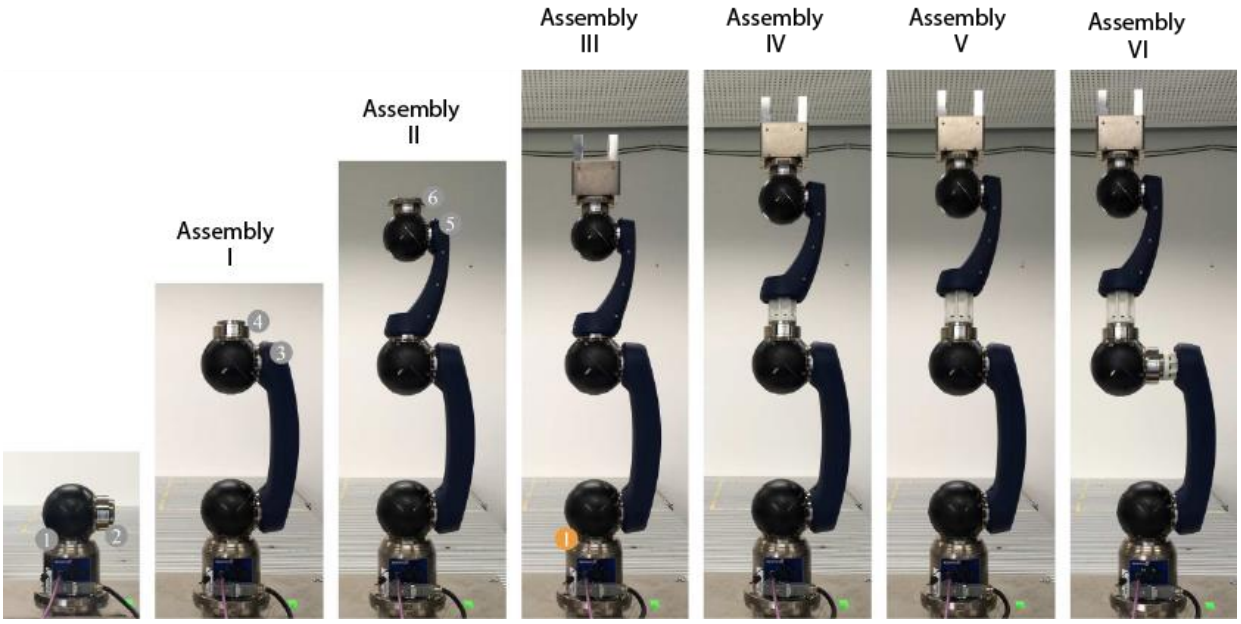

Fig. S9. Assemblies used for tests.

For validation, we use the assemblies from Fig. S9. The results of these tests are collected in Fig. S10 showing the comparison between the measured and the predicted torque. Overall, these plots show good model matching and demonstrate the effectiveness of our automatic model generation.

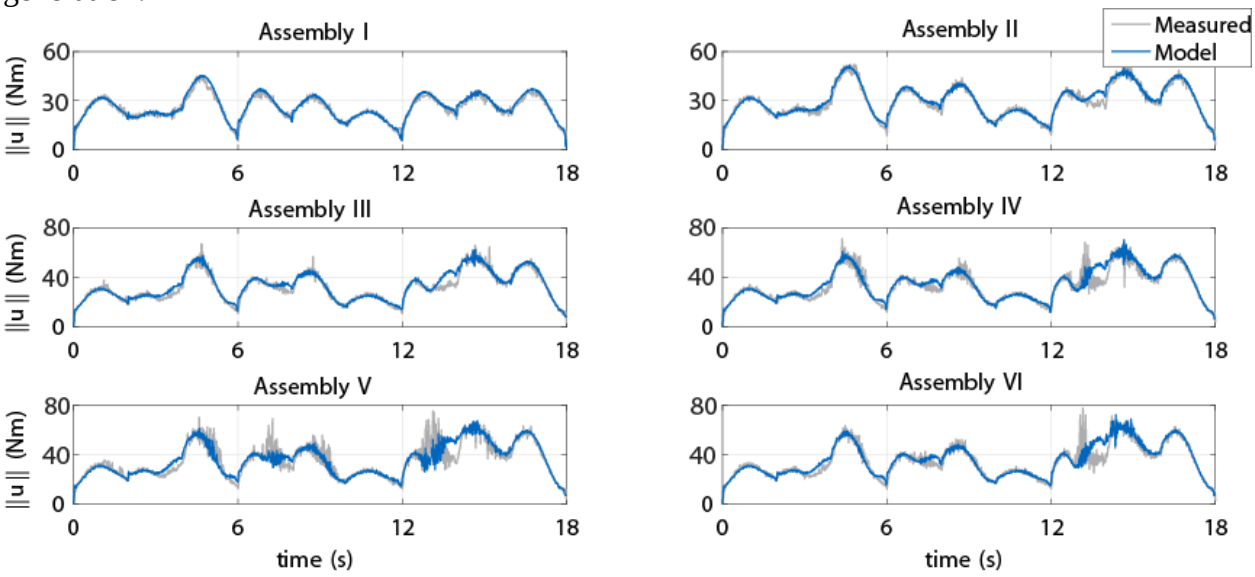

Fig. S10. Experimental verification of the automatically generated models.

Evaluation of the Control Performance

We validate the tracking performance of different joint space controllers using the same trajectories and assemblies as in the previous model validation. The results of these experiments are presented in Fig. S11, which shows remarkable insensitivity of the tracking error with respect to changes of the assembly. Fig. S11 also shows that passivity-based control performs better than

inverse dynamics control after careful tuning of the control parameters. The fair tuning is confirmed by similar measurement noise amplification in the torque commands as shown in Fig. S12.

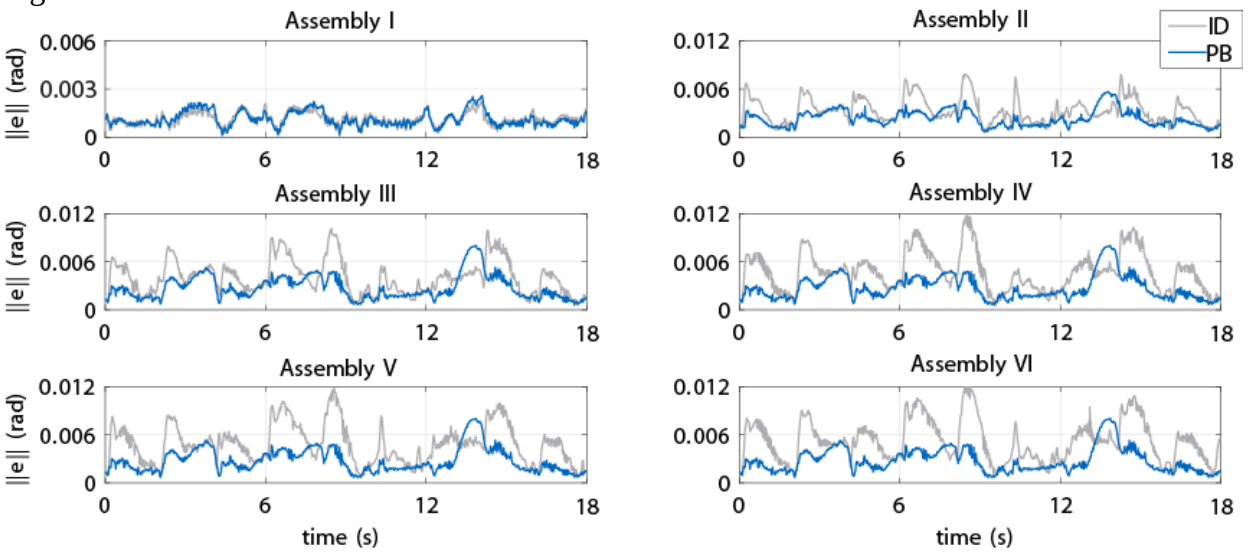

Fig. S11. Tracking performance comparison when using inverse dynamics control (ID) and passivity-based control (PB), for different assemblies.

The performance of the adaptive friction compensation is tested by a badly-tuned version of the automatically generated passivity-based controller and by setting all friction parameters initially to zero. The results of this test in Fig. S13 show a reduction of the norm of the tracking error over time. In this experiment, the following gains have been used: $\mathbf{K}_V = \mathbf{K} = 30\mathbf{I}$, $\mathbf{K}_{\Delta_\beta} = \mathbf{I}$.

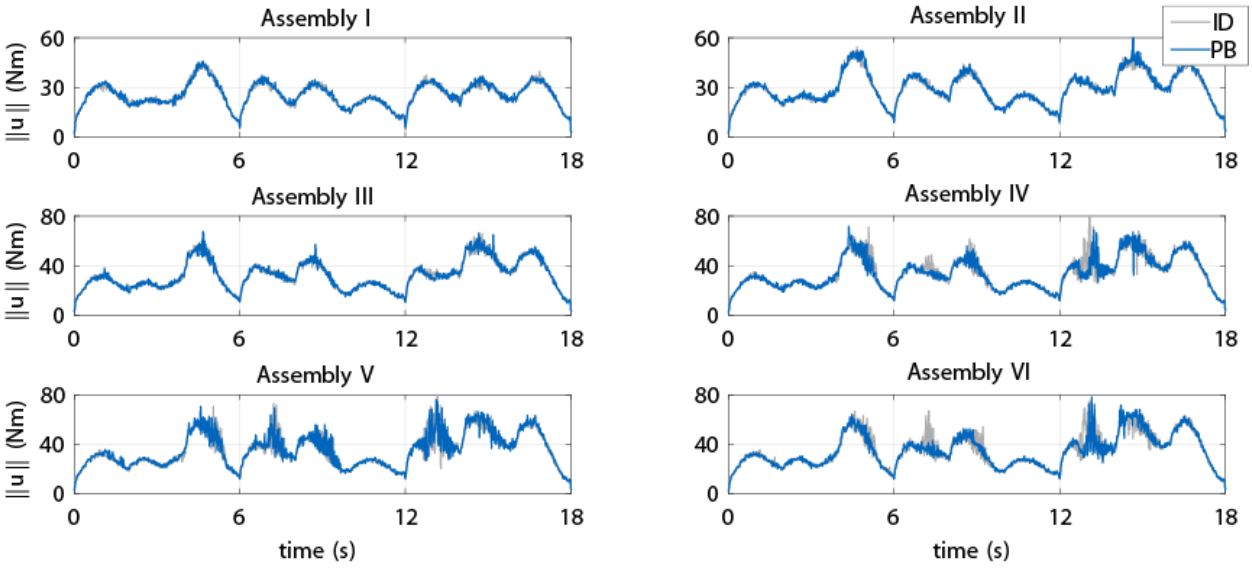

Fig. S12. Control torque commands required when using inverse dynamics control (ID) and passivity-based control (PB) for the trajectory tracking test.

An additional performance comparison is presented in Fig. S14 to show the benefit of our self-programmed controllers to a model-free approach. As a model-free approach, we use a distributed proportional-integral-derivative (PID) controller that is implemented in each joint separately. This performance comparison has been executed for assemblies I, II, and III of

Fig. S9. The gains of the controllers have been selected by setting $\omega_n = 50(rad/s)$ and $\zeta = 0.65$ for both inverse-dynamics control and the PIDs. The passivity-based control law with adaptive friction compensation has been tuned such that the tracking performance with assembly I is comparable to the other controllers: $\mathbf{K} = 50\mathbf{I}$, $\mathbf{K}_V = 50\mathbf{I}$, $\mathbf{K}_{\Delta_\beta} = \mathbf{I}$. While the tracking error when using assembly I was still satisfactory for the PID controllers, their performance dramatically decreased when changing the robot assembly without retuning as shown in Fig. S14. Also, instabilities were observed for the decentralized PIDs when increasing the degrees of freedom for the robotic assembly; however, the automatically generated model-based controllers show a remarkable insensitivity.

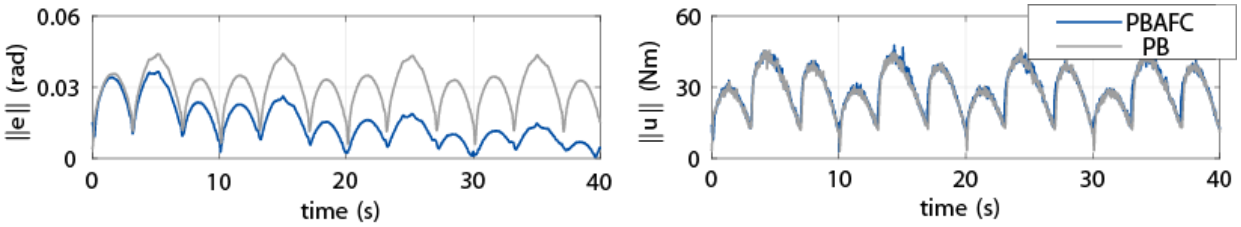

Fig. S13. Demonstration of the adaptive friction compensation. PB denotes the standard passivity-based controller, and PBAFC denotes the passivity-based control law with adaptive friction compensation (*54*).

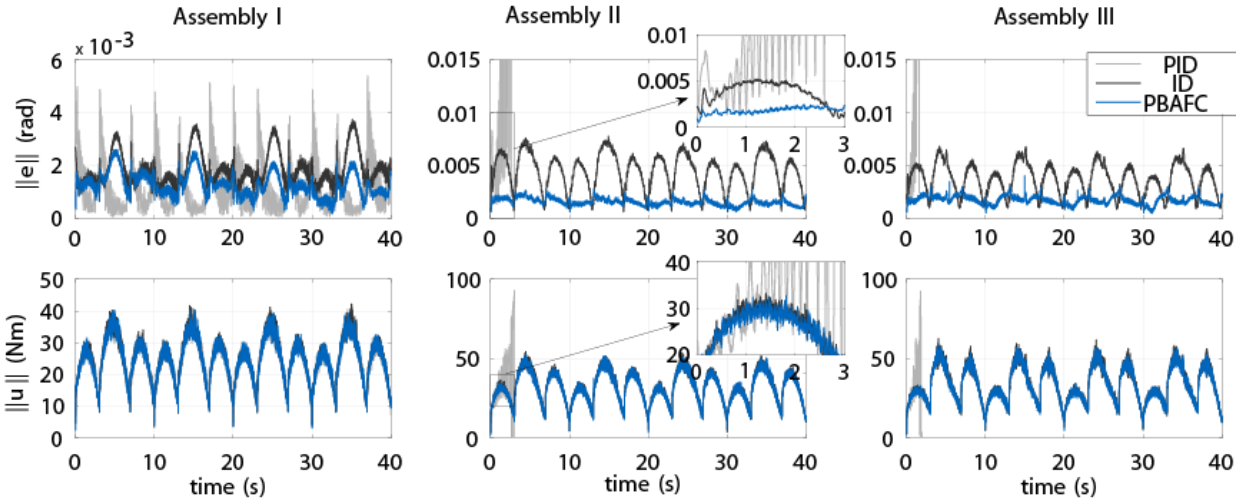

Fig. S14. Experimental comparison of different controllers when changing the assembly of the robot. ID denotes the inverse dynamics controller, PBAFC the passivity-based control with adaptive friction compensation, and PID the scheme with decentralized PID control (*54*).

A video showing the complete architecture of Fig. S6 in action is attached.

Conformance Checking of the Full Body Model

We have validated our full body model using reachset conformance checking (*96*). A model is said to be reachset conformant if all measurements lie within the reachable set of the model. As measurements we have used the data from the publicly available motion-capture database from

Carnegie Mellon University Graphics Lab (mocap.cs.cmu.edu, accessed 28.02.19). The movements we use are grouped into four categories:

- Everyday motions, e.g., construction work, machining work, manipulating objects, stumbling (96 files).
- Sports-related motions, e.g., boxing, throwing/shooting balls and batting balls (67 files).
- Dance-related motions, e.g., swing dance, Indian dance and modern dance (58 files).
- Acrobatic motions, i.e., any motion where both feet are in the air simultaneously, including jumps, cartwheels, backflips, and swings from a trapeze (68 files).

The everyday motions are expected to be similar to usual movements in a human-robot co-existence scenario. We also consider sports-related, dance-related and acrobatic movements. For the reachset conformance checking we have used the MATLAB toolbox CORA (*103*).

For each marker we have assumed the measurement uncertainty to be $\delta\mathbf{y} = 0.04$ [m] for the position and $\delta\dot{\mathbf{y}} = 0.4$ [m/s] for the velocity. The data has been recorded with either 60 or 120 frames per second. For each frame and each movement, we have computed the reachable set for a prediction horizon of 0.2 [s] since larger horizons did not return different results due to the quick expansion of reachable sets and since in that time any extremity of the human can be fully extended.

For the full body we only use the task-space approach and only for the arms we additionally use the joint-space approach as arms have the biggest influence on human-robot co-existence. Since conformant parameters for the arms are already reported in (*73*), we only present the maximum velocity $v_{y,max}$ and the maximum acceleration $a_{y,max}$ required for the task-space approach. The maximum velocity $v_{y,max}$ is similar for many body parts; this is intuitively clear since almost all body parts have similar velocities when running. On the contrary, not all body parts have the same maximum accelerations, e.g., the torso cannot accelerate as fast as arms or legs. Some acrobatic and sports movements cause ground impacts. Since impacts cause very high accelerations, we have excluded impact accelerations, since those accelerations do not contribute to accelerating in free space, but are preventing body parts from penetrating the ground.

The maximum acceleration for different body parts which ensures reachset conformance is presented in Table S2 and the markers belonging to the various body parts are shown in Fig. S15. In terms of the velocity model, $v_{y,max} = 14$ [m/s] for all body parts establishes reachset conformance.

Table S2. Maximum acceleration of different body parts. The positions of the markers are shown in Fig. S15.

Body part	**Markers**	$a_{y,max}$ **in [m/s²]**
Torso	LFWT, RFWT, LBWT, RBWT, CLAV, STRN, C7, T8, T10, NEWLBAC, RBAC, NEWRBAC	20
Head	LFHD, RFHD, LBHD, RBHD	25
Arm and hand	LSHO, NEWLSHO, LUPA, LELB, LFRM, LWRA, LWRB, LFIN, LTHMB, RSHO, NEWRSHO, RUPA, RELB, RFRM, RWRA, RWRB, RFIN, RTHMB	50
Thigh and knee	LTHI, LKNE, RTHI, RKNE	30
Shin and foot	LSHN, LANK, LHEE, LMT1, LMT5, LTOE, LRSTBEEF, RSHN, RANK, RHEE, RMT1, RMT5, RTOE, RRSTBEEF	50

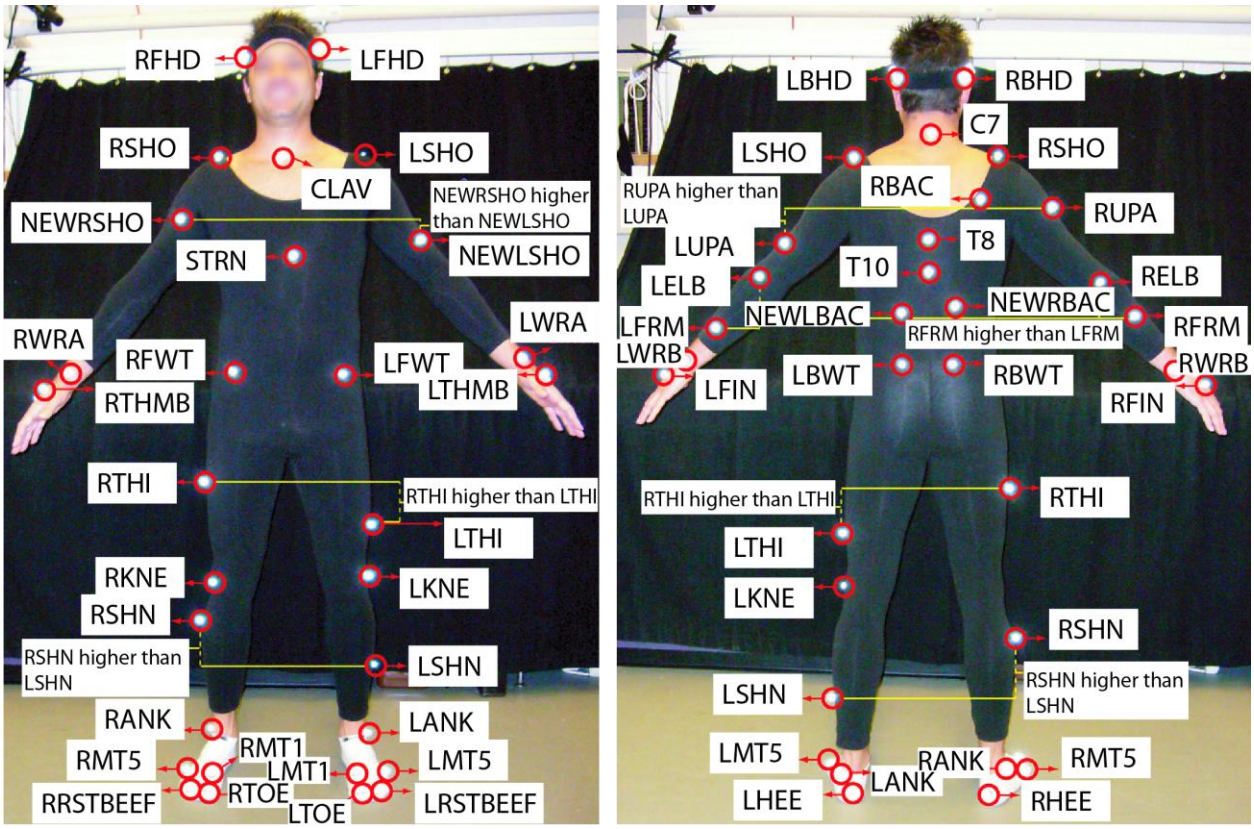

Fig. S15. Markers of the motion-capture database from Carnegie Mellon University Graphics Lab (photos taken from mocap.cs.cmu.edu, accessed 28 February 2019).

Experimental Data of the User Study

In this subsection, we describe the user study presented in the paper in more detail. While the average idle time of the robot and the average time to completion of the human are presented in the paper, we additionally show the box and whisker plots for the first four trials and the last four trials in Fig. S16. In addition to measuring the idle times and the time to completion, we evaluated the following four hypotheses:

- **Hypothesis 1**: Idle time of the robot is significantly different for self-verification and verification with static safety zones for a) the first four trials, b) the last four trials, and c) all trials.

  Test: For each subject, the mean value of the idle time was calculated for a) the first four trials b) the last four trials and c) all trials. Distribution of robot idle time over subjects is significantly different from the normal distribution in the first four trials for static safety zones. Kruskal-Wallis test used: $p < 0.0001$. The results for the different trials are

  - Set of trials a): Median robot idle time is 37.7% lower for self-verification than for static safety zones;
  - Set of trials b): Median robot idle time is 36.8% lower for self-verification than for static safety zones;

- Set of trials c): Median robot idle time is 36.0% lower for self-verification than for static safety zones.

Hypothesis confirmed.

- **Hypothesis 2**: Time to completion of the human is significantly different for self-verification and static safety zones, for a) the first four trials, b) the last four trials, and c) all trials.

  Test: For each subject, the mean value of time to completion was calculated for a) the first four trials b) the last four trials and c) all trials. Distribution of human time to completion over subjects is significantly different from the normal distribution in the last four trials using static safety zones. Kruskal-Wallis test used: $p > 0.05$; no significant difference. **Hypothesis rejected**.

- **Hypothesis 3**: Idle time of the robot decreases with the number of trials, using a) self-verification and b) static safety zones.

  Test: Distribution of mean robot idle time averaged over all trials is not significantly different from the normal distribution. Repeated-means ANOVA test used: $p < 0.001$. The results for the different trials are

  - Set of trials a): Decrease in mean idle time over 12 trials of the robot using self-verification is 14.0%,
  - Set of trials b): Decrease in mean idle time over 12 trials of the robot using static safety zones is 11.3%.

  Hypothesis confirmed.

- **Hypothesis 4**: Time to completion of the human decreases with the number of trials, for a) self-verification and b) static safety zones.

  Test: Distribution of mean human time to completion averaged over all trials not significantly different from normal. Repeated-means ANOVA test used: $p < 0.001$. The results for the different trials are:

  - Set of trials a): Decrease in mean time to completion over 12 trials of the human using self-verification is 22.5%,
  - Set of trials b): Decrease in mean time to completion over 12 trials of the human using static safety zones is 24.9%.

  Hypothesis confirmed.

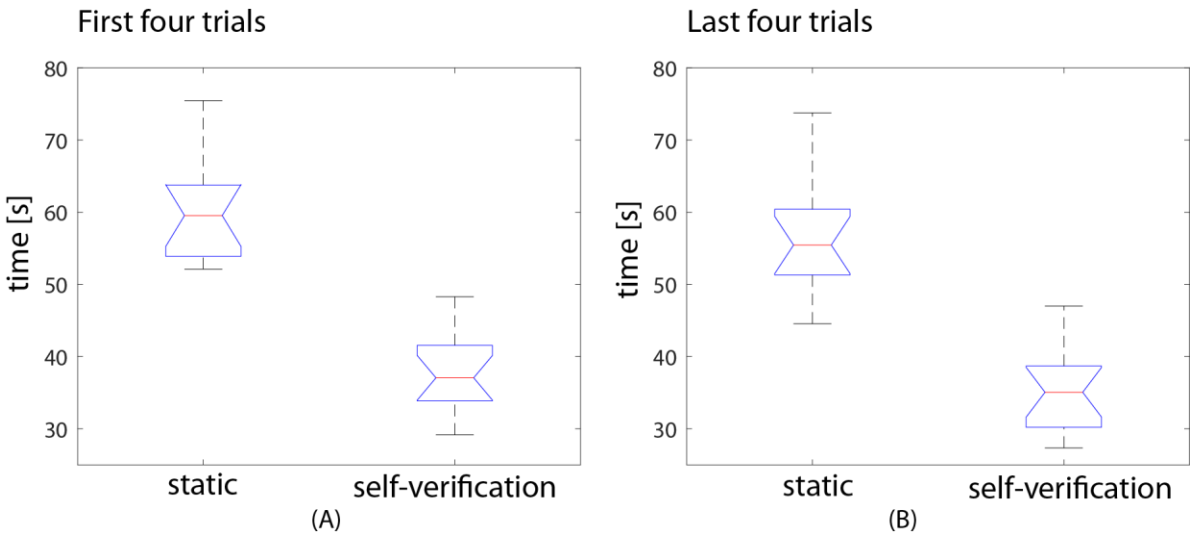

Fig. S16. Box and whisker plot of robot idle time. (A) First four trials. **(B)** Last four trials.

Software for optimal module composition

The complete software for determining the optimal module composition, computing the cost function of each module composition, and computing the cost function of the competing robots in the experiment (Schunk LWA 4p, Kuka LWR 4+. Stäubli TX90) is provided in the data file S2 under "/OptimalComposition" ("main_optimalComposition.m" and "otherRobots.m") and requires MATLAB R2018a. The code has been written by Stefan Liu and Esra Icer (esra.icer@tum.de).

For computing the cost function of all robots, their kinematic, dynamic, and geometric model are needed. The model of IMPROV and the standard configured Schunk LWA 4P have been obtained using identification experiments. The models of the KUKA and Stäubli robot have been obtained from (*104-106*) and websites[2]. The models for all robots are also provided within the repository.

Movie S1. Interplay of self-programming and self-verification.

This video shows the whole process from the assembly of robots from modules to self-programming and self-verification. The first part of the video presents the set of used modules and how their dynamic and kinematic properties as well as their geometry are utilized for self-programming. Next, the video shows a reconfiguration process and how the data in each module is collected after reconfiguration. The second part of the video explains the self-verification principle based on predicted occupancies of humans. Presented experiments for two different robot compositions with six degrees of freedom (Base–PB1–L1–PB1–L2–PB2–Gripper) and eight degrees of freedom (Base–PB1–L3–PB1–L4–PB1–L2–PB2) show that the robot stops in

[2] https://github.com/CentroEPiaggio/kuka-lwr and https://www.staubli.com/en/robotics/customer-support/cad-library/

time before the human reaches it. The geometric model needed for the collision checking of both compositions is self-programmed using the data from each module.

Movie S2. Comparison of optimal IMPROV compositions with commercial robots.
This video shows the experiment described in Fig. 7 of the paper and the results in Table 1 of the paper, which compares the performance of IMPROV modules to the commercially-available Schunk LWA 4p, KUKA LWR 4+, and Stäubli TX90. For both tasks, a simulation of the trajectories of each robot is shown, as well as the obstacles.

Data file S1. Robot hardware.
STL files of 3D-printed modules, and technical specification of robot and controller used for experiments. The provided modules enable one to recreate the modular robot presented in this paper.

Data file S2. Software.
Code for self-programming and optimal module composition executable on MATLAB R2018a. The code for self-programming creates kinematic, dynamic, and geometric models for composed modules. The software for optimal compositions includes software for task specification, collision checking, and path planning. Furthermore, software for visualization of composed modules is provided.

Data file S3. User study evaluation.
This code can be executed on MATLAB 2016b and requires the statistic toolbox. All code is from Aaron Pereira, except the functions "cronbach.m" and "swtest.m", which were written by Alexandros Leontitsis and Ahmed ben Saida, respectively. The file you want to execute is "test_hypotheses.m". Everything else is self-explanatory or explained in the functions themselves.