From Composable Models to Correct-by-Construction Software for Contact-Rich Robotic Mobile-Manipulation Tasks

Sven Schneider, Vamsi Kalagaturu, Herman Bruyninckx, and Nico Hochgeschwender, *Member, IEEE*

***Abstract*—Software frameworks like the *Stack of Tasks* (SoT), the Stanford *Whole-Body Control* (WBC) library, or the *instantaneous Task Specification using Constraints* (iTaSC) have enabled robots to perform advanced, contact-oriented manipulation tasks. *jgeom_constr* and *eTaSL* are among the few formal, computer-interpretable languages that allow users to specify such tasks independent of these frameworks. We analyse these languages for their limitations with respect to *composability*, the design for extensibility without having to change existing models, and *compositionality*, meaning that the semantics of compositions unambiguously follows from the semantics of the components and of the composition relations. To overcome these limitations we design a graph-structured and well-defined interchange format for such tasks. The associated tooling enables us to generate correct-by-construction code that adheres to predefined rules and constraints. We showcase our models and toolchain by *incrementally* constructing a workspace-alignment application for a highly-redundant mobile platform that is equipped with two 7-DoF, torque-controlled manipulators.**

***Index Terms*—Software tools for robot programming, software, middleware and programming environments, multi-contact whole-body motion planning and control, mobile manipulation, behavior-based systems.**

I. Introduction

THE *execution* of contact-rich manipulation tasks as shown in Fig. 1 is the forte of constraint-based task execution frameworks such as the Stack of Tasks (SoT) [1], the Stanford Whole-Body Control (WBC) library [2], ControlIt! [3], or the instantaneous Task Specification using Constraints (iTaSC) [4], [5]. Here, a *task* refers to a desired robot behaviour or motion that is explicitly defined by a set of constraints, such as bilateral constraints on the distance relations shown in Fig. 1, which are monitored or enforced during the execution. All of these frameworks offer custom application programming interfaces (APIs) to represent such tasks. On the one hand, these APIs are designed for human consumption and remain inaccessible to reason about by a robot. On the other hand, they interleave the task specification with implementation details about the respective framework or programming language and, hence, prevent reusing specifications between the mutually incompatible frameworks.

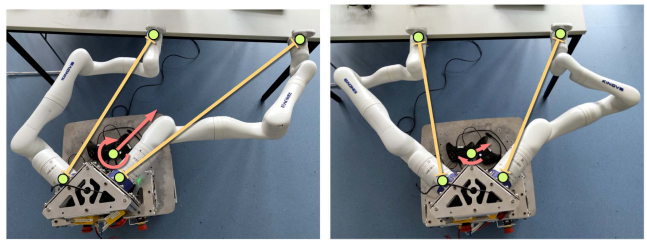

Fig. 1. Scenario *SC1* shows a wheeled mobile manipulator aligning in front of a table while using its compliant arms as "whiskers" to estimate its position and orientation with respect to the table's edge. The task is specified by constraining the distances between the arms' shoulders and the contact points with the table (yellow) to equal reference values. Controlling the mobile base with the derived force and torque (red) satisfies the constraints.

To address some of these problems, the JSON-based geometric constraint expressions *jgeom_constr* [6] and *eTaSL* [7] provide domain-specific languages (DSLs) for users to *declaratively* specify tasks. We have selected the former as a representative (Listing 1). In the following we indicate model elements typographically by a `monospaced font`. The listing composes a task from three, possibly empty, sets of constraints (`primary`, `auxiliary` and `safety`) and a collection of `monitors`. A constraint consists of an `expression` (a linear or angular distance between points, lines or planes) as well as a `behaviour` that represents a trajectory and an associated proportional controller with the gain `specification`. The `monitor` contains another expression that must be greater than (`more`) the threshold (`bound`) to raise an event. We analyse these task-specification models to identify structural and semantic anti-patterns, as we previously did for kinematic chain specifications in [8].

Received 31 March 2025; accepted 21 July 2025. Date of publication 11 August 2025; date of current version 20 August 2025. This article was recommended for publication by Associate Editor D. Papageorgiou and Editor M. Vincze upon evaluation of the reviewers' comments. The work of Sven Schneider was supported by the Bonn-Aachen International Center for Information Technology. The work of Vamsi Kalagaturu and Nico Hochgeschwender was supported by SOPRANO, Horizon Europe under Project 101120990. The work of Herman Bruyninckx was supported by RobMoSys (H2020-732410, Composable Models and Software for Robotics Systems-of-Systems), Esrocos (H2020-730080, European Space Robot Control Operating System). This work was supported by European Unions Horizon 2020 Project SESAME (H2020-101017258, Secure and Safe Multi-Robot Systems). *(Sven Schneider and Vamsi Kalagaturu are co-first authors.) (Corresponding author: Vamsi Kalagaturu.)*

Sven Schneider is with the Department of Computer Science, Institute for AI and Autonomous Systems, Hochschule Bonn-Rhein-Sieg, 53757 Sankt Augustin, Germany, and also with the Department of Mechanical Engineering, KU Leuven, 3000 Leuven, Belgium (e-mail: sven.schneider@phd.h-brs.de).

Vamsi Kalagaturu and Nico Hochgeschwender are with the Department of Mathematics and Computer Science, University of Bremen, 28359 Bremen, Germany (e-mail: vamsi.kalagaturu@uni-bremen.de; nico.hochgeschwender@uni-bremen.de).

Herman Bruyninckx is with the Department of Mechanical Engineering, KU Leuven, 3000 Leuven, Belgium, and with the Department of Mechanical Engineering, TU/e Eindhoven, 5612 Eindhoven, AZ, The Netherlands, and also with Flanders Make, 3920 Lommel, Belgium (e-mail: herman.bruyninckx@kuleuven.be).

Digital Object Identifier 10.1109/LRA.2025.3597864

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Listing 1: Excerpt of a JSON model of a *jgeom_constr* task specification that moves the robot's right shoulder to the right end-effector. The bold green font represents DSL keywords while the pink font represents model instance data.

```
{ "type": "geom_task",
  "task_name": "align_to_workspace",
  "auxiliary": [], "safety": [], "primary": [ {
    "type": "constraint",
    "constraint_name": "c_dist_rightarm_shoudler_ee",
    "behaviour": {
      "type": "behaviour",
      "behaviour_type": "positioning",
      "specification": 2,
      "traj_gen": {
        "type": "trajectory",
        "traj_type": "trapezoidal_with_duration", ... }},
    "expression": { "expression": {
      "type": "geometric_expression",
      "expression_type": "point_point_distance",
      "expression_name": "shoulder_ee_distance",
      "primitive_1": {
        "entity": {
          "type": "point", "x": 0, "y": 0, "z": 0 },
        "reference_frame": "rightarm_shoulder" },
      "primitive_2": { ...
        "reference_frame": "rightarm_ee" } } } } ],
  "monitors": [ {
    "type": "monitor",
    "monitor_name": "rightarm_shoulder_ee_distance",
    "comparison_type": "more",
    "monitor_var_type": "pos",
    "bound": 0.6,
    "event_risen": "e_rightarm_shldr_ee_dist_crossed",
    "monitored_expression": { ... } } ] }
```

AP1) Underutilized identifiers and incomplete metamodels: *The* key for composability is the ability to refer to, or address, entities via *symbolic pointers*, also from external models. However, *jgeom_constr*'s JSON-Schema[1] metamodel represents only *composition* relations, not graph-structured *aggregations*, thus lacking explicit identifiers. The `_name` properties and symbolic pointers (e.g. `reference_frame`) remain mere strings. *jgeom_constr* ignores these identifiers, forcing duplication of identical expressions across behaviours and monitors. Additionally, rather sooner than later one wants to integrate the task specification with an encompassing system model. While this integration model should indeed not be part of the task specification, it should still *explicitly*, via unique identifiers and the complementary symbolic pointers, establish the connection to (i) a constraint controller and a solver; (ii) the `reference_frames` in the underlying kinematic chain model; as well as (iii) the run-time variables (`primitives`) that are derived from sensory measurements. Finally, we also see that all numeric values lack units of measurement. As a result of these omissions, a computer is unable to fully and unambiguously interpret these models.

AP2) Artificial structural constraints jgeom_constr relies on easy to parse and process tree-structured models. As a consequence, there exists a single root element, the `geom_task`, that artificially limits the composition hierarchy to the top. Furthermore, the models (i) can only represent proportional controllers; and (ii) do not distinguish between the *definition* (the identity and type) and the *representation* of geometric primitives. Lastly, the constraints in a task are classified as `safety`, `primary` and `auxiliary`. Even if this is a common, three-tiered scheme, the complete model is that of *relative priorities* between a set of constraints

AP3) Tight coupling deep in composition hierarchy: The previous anti-pattern is aggravated by the high variability deep in the composition hierarchy. For instance, a wide range of controllers could realize a `behaviour`, yet only a linear controller is supported here. Such choices propagate as dependencies to the root, so that any change or extension entails a new version of the DSL. As a result the composition hierarchy fans out to the bottom, i.e. the single top layer depends on the many concrete details of the lower layers. For better composability, a fan out to the top is instead desirable, resulting in a hierarchy of (i) a distance relation; (ii) a constraint (equality, inequality or bilateral) on that distance together with a reference value or threshold; (iii) a controller that enforces the constraint; and (iv) optionally a trajectory to change the reference values or thresholds.

AP4) Implicit assumptions: The `geometric_expressions` implicitly interleave (i) the definition and representation of a distance; and (ii) its computation given the coordinates of further geometric entities (the `primitive_1` and `primitive_2`). Only the former should exist as an entity in the *declarative* motion specification models, whereas the latter should only enter in an overall system model. Furthermore, the primitives only represent the coordinate model, but lack complete coordinate-free geometric models as pioneered in the Geometric Relations Semantics [9]. Such coordinate-free models can not only uncover common specification errors, they also establish the symbolic links between the task specification and the world model or the robot model.

The problem that originates from these anti-patterns is that they impede the automatic and unambiguous interpretation of models such as these motion specifications. This leads to bugs that are difficult to identify and trace, both, in the specifications themselves but also in the associated software tools. We overcome the identified anti-patterns by (i) designing composable and compositional models for contact-rich motion specifications; (ii) automatically verifying (yet *not* validating) these models; and (iii) generating correct-by-construction code given such models. Such rigorous specifications are especially required in high-assurance and safety-critical robots where it is crucial to prevent bugs from occurring in the first place.

The contributions of this letter are as follows:

- We *analyse* existing DSLs for specifying contact-rich robotic tasks. This analysis also supports the underlying analysis methodology from our prior work.
- Given that analysis we *design* and *develop* graph-structured, composable and compositional (meta) models that enable us to connect the task specifications to the overall robotic system including the kinematic chains together with their kinematics and dynamics solvers.
- We *implement* the tooling to generate correct-by-construction code from the models.
- We *exploit* the models and tools for specifying and realizing a contact-rich alignment task on a real-world, highly-redundant mobile manipulator.

After this introduction and analysis, in Section II we present our composable and compositional (meta) models. Section III describes the required software tooling, while Section IV presents the results for two scenarios. In Section V we discuss the results to then summarize the paper in Section VI.

[1] https://json-schema.org/draft/2020-12/json-schema-core.html

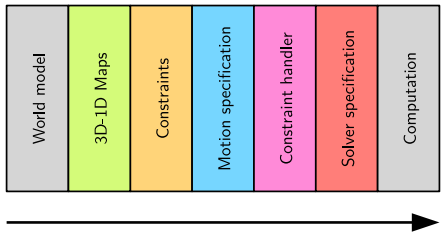

Fig. 2. Model composition workflow starts on the left and proceeds by introducing relations towards the right. In this letter we assume the world model — the kinematic chain — to be given and keep the computational model implicit.

II. Composable Models for Compliant Interaction Tasks

In this section we revisit task specification languages to derive metamodels that enable us to define and verify concrete, composable models. To this end we rely on JSON-LD[2] as graph-based modelling language in conjunction with SHACL[3] for describing structural model constraints. We have made the following models and an in-depth tutorial available online under free and open-source licenses via:

https://github.com/secorolab/motion-spec-ral
https://secorolab.github.io/motion-spec-ral

A. From DSLs to Composable (meta) models

We observe that *user-facing* DSLs in general, intentionally *limit* the users' choices, meaning they only expose a subset of the possible model choices. However, rather sooner than later, developers want to "connect" (or rather *compose*) such models to an overall application. Then the issues above prevent the automatic processing (e.g. composition into an application, verification or code generation) of these models by a computer. It is important to note that all of the concepts in the above DSLs *are* relevant for task specification languages, but it is the *representation* of these concepts that we improve. Namely, we strive for models that are (i) *composable* so that it is always possible to extend them; and (ii) *compositional* so that the models are free from implicit choices or hidden assumptions, ready to be unambiguously interpreted by a computer.

Hence, in [8] we proposed the application of JSON-LD — one of the few standardized graph interchange formats — in the context of robotics. We also employ it here to present concrete models, that conform to our metamodels, along the scenario depicted in Fig. 1. The robot's objective is to keep its end-effectors in contact with the table's edge while aligning the base (i) parallelly to that edge; and (ii) centrally between both contact points. Geometrically, this situation can be achieved when the left and right shoulder-contact distances are equal, while the distance between the two contact points differs from the distance between the robot's shoulders.

Fig. 2 depicts the relevant domains and the order in which to compose models from these domains. In the following we discuss the five intermediate domains — maps between 3D and 1D spaces, the constraints, the motion specification, the constraint handlers, and the solver specification. For each of these we have identified and implemented (meta) models with the correct granularity of primitives and relations, i.e. each composition layer adds exactly one entity that in itself is minimal, introduces no implicit assumptions, while keeping the overall model in a coherent and consistent state. The world model includes the kinematic description of the robot, which we presented in [8], and can also include environmental features such as the table. The *imperative* computational domain remains out-of-scope and will implicitly recur as template-based code generation. A complete computational model will include concepts such as algorithms, threads, or processes that implement the *declarative* task specifications. In [10] we have presented the utility of such models.

[2]https://www.w3.org/TR/json-ld/
[3]https://www.w3.org/TR/shacl/

Listing 2: Model of a distance relation between two points.

```
1 { "@context": {
2     "geo": "https://comp-rob2b.github.io/...#",
3     "LinearDistance": "geo:LinearDistance",
4     "PointToPointDistance": "geo:PointToPointDistance",
5     "between-entities": {
6       "@id": "geo:between-entities",
7       "@type": "@id" } },
8   "@id": "dist-rightarm-shoulder-ee",
9   "@type": [ "PointToPointDistance", "LinearDistance" ],
10  "between-entities": [
11    "rightarm_shoulder_origin", "rightarm_ee_origin" ] }
```

Listing 3: SHACL shape constraints for distance relations.

```
1 @prefix geo: <https://comp-rob2b.github.io/...#> .
2 geo:LinearDistance a rdfs:Class, sh:NodeShape ;
3   sh:property [
4     sh:path geo:between-entities ;
5     sh:minCount 2 ; sh:maxCount 2 ; sh:nodeKind sh:IRI ] .
6 geo:PointToPointDistance a rdfs:Class, sh:NodeShape ;
7   sh:property [
8     sh:path geo:between-entities ; sh:class geo:Point ] .
```

B. Geometric Primitives, Relations and Their Coordinates

Structurally, a JSON-LD model as exemplified in Listing 2 consists of a JSON object that contains `@`-prefixed JSON-LD keys. The `@context` is part of the metamodel and defines the meaning of keys (or terms) or types (`@type`) by mapping them to unique Internationalized Resource Identifiers (IRIs). This mapping either takes the form of an "alias" as shown in lines 3 and 4 or using the `@id` key in the context as shown in line 6. For terms it can also determine the associated values' types as shown in line 7 that defines `between-entities` as a symbolic pointer. All these mappings rely on the prefix `geo` defined in line 2 to compact IRIs. The `@id` and `@type` keywords recur in the model (lines 8 and 9) to define the entity's identifier and one or more types, respectively (cf. AP1). These types are also the entry points for the two SHACL shape constraints exemplified in Listing 3. The first shape constraint applies to the `LinearDistance` type and enforces that the `between-entities` property references exactly two entities, whereas the second shape constraint, that applies to `PointToPoint-Distances`, restricts these entities to be of type `Point`. Hence, we note that SHACL shape constraints are also composable.

Semantically, this model describes a distance relation between the two points referenced by the `between-entities` term, that are not detailed further in this letter. Listing 4 describes one

Listing 4: Coordinate representation for Listing 2.

```
1 { "@id": "dist-coord-rightarm-shoulder-ee",
2   "@type": [ "DistanceReference", "DistanceCoordinate" ],
3   "of": "dist-rightarm-shoulder-ee",
4   "unit": "M" }
```

Listing 5: Model of a computation.

```
1 { "@id": "dist-coord-rightarm-shoulder-ee",
2   "@type": [ "DistanceReference", "DistanceCoordinate" ],
3   "of": "dist-rightarm-shoulder-ee",
4   "unit": "M" }
```

of possibly many coordinate models that references the previous coordinate-free distance relation (line 3) and is measured in metres (line 4). As an alternative we also foresee a distance model that fuses the coordinate-free and the coordinate representation. Distances are invariant with respect to the choice of coordinate frame and hence do not reference a coordinate frame in contrast to geometric relations such as positions, velocities or accelerations (cf. AP4). Further linear distances between point, line or plane primitives and angular distances between line or plane primitives follow the same structure but differ in their semantics as encoded by the type. All of the above entities and relations originate from the world model or the robot model.

C. Mappings Between 3D and 1D Quantities

Many tasks can be described in a low-dimensional configuration space. The distance between the robot's shoulder and the contact point with the table is one of many examples where even a 1D task description suffices. It is then the composition of multiple such descriptions that shapes the robot's overall, purposeful motion. Consequently, we need ways to map from 3D Cartesian quantities to such 1D quantities and back again. The first option is to establish a (passive) *view* into an existing model that establishes a relation between two independent quantities: one takes the role of a *source* that lives in the 3D Cartesian space, and the other the role of a *target* living in a 1D subspace. Both of these quantities must be type- and unit-compatible. Software analogies of such views include pointers and indexing into an array or a data structure. The second option is to introduce an (active) *operator* or computation. One example of such an operator that we employ in this letter comprises the computation of a distance coordinate given the forward position kinematics of a kinematic chain. Another example is shown in Listing 5, a model that maps a 1D force `magnitude` f (e.g. a controller's output) to the `wrench` ${}_{[a]}\boldsymbol{F}^{b}$ measured at point b and expressed in frame $[a]$'s coordinates, given the `direction` vector $\boldsymbol{d}$ and the `position` vector ${}_{[a]}\boldsymbol{r}^{b,c}$ from the measurement point b to the point of force application c. Formally, this operator is represented by Eq. (1).

$$ {}_{[a]}\boldsymbol{F}^{b} = \left({}_{[a]}\boldsymbol{r}^{b,c} \times \left(f \cdot {}_{[a]}\boldsymbol{d} \right) \quad f \cdot {}_{[a]}\boldsymbol{d} \right) \tag{1} $$

Our explicit model enables the verification of structural constraints that the formula only implies by recurring letters.

D. Constraint Relations and Motion Specifications

The next composition step is to impose a desired state on such scalar quantities. This is achieved by a constraint relation between the to-be-constrained quantity and (i) a reference value for an equality constraint; (ii) a threshold for *greater than* or *less than* inequality constraints; as well as (iii) a lower and an upper threshold for a bilateral constraint. The referenced quantity must be compatible in type and in unit with the thresholds.

A guarded motion specification composes multiple such constraints that *together* define the intended behaviour of a robot over some time horizon. Here, the "guard" refers to the post-conditions, that is, a set of constraints *any* of which can terminate the motion [11], [12]. Examples of such criteria comprise a timeout, reaching a destination or exceeding a force threshold (cf. AP2). These criteria are complemented by symbolic pointers to the pre-conditions — constraints that *all* must hold before the motion can start — and the per-conditions that must be enforced during the motion.

E. Constraint Evaluators, Monitors and Controllers

The previous models are a declarative specification of a motion: by design they lack any behaviour. The first model of such a behaviour is the constraint evaluation which in most cases represents the computation of an error. The error signal is an additional quantity referenced by the constraint evaluation model, which is then consumed by a constraint monitor or a constraint controller (cf. AP3). A *monitor* introduces behaviour by signalling the motion's state to the time-discrete coordination logic including state machines or Petri nets [13] and comes in two shapes: a level-triggered monitor and an edge-triggered monitor. The former signals the constraint's value by raising a *flag* when the constraint gets satisfied and lowers it again in the opposite case. The latter signals a change of the constraint and can be configured to emit an *event* on the rising or falling edge. A *controller* introduces behaviour by shaping the time-continuous behaviour of the kinematic chain. To this end it maps the error to a control signal. The model of a controller introduces specific parameters such as the controller gains. Here, we rely on 1D PID controllers and variations thereof. The control signals are mapped to 3D Cartesian space as explained above.

F. Solver Specification

Instantaneously the control signals must be mapped to the robot's actuator while resolving potential conflicts between these signals. We call these instantaneous solver inputs *motion drivers* because they are responsible for the robot's purposeful motion, each serving a dedicated solver interface: feed-forward joint-force and external Cartesian forces exerted on joints and links, respectively, as well as acceleration constraints imposed on certain links. Here, we also introduce the prioritization hierarchy between all motion drivers. The attachment of the latter two motion drivers to specific links—along with their associated coordinate frames—forms the basis for (i) computing the Jacobian matrix or the operational-space inertia matrix for solvers that rely on pseudoinverses or on quadratic programming; as well as (ii) injecting the associated quantity at the correct link for recursive solvers. The top-level composition model connects the motion drivers to a particular kinematics or dynamics solver. This model also comprises the solver's configuration, including the requested outputs, the selected kinematic chain and a root frame from which to start the traversals over that chain, as well as a gravity model.

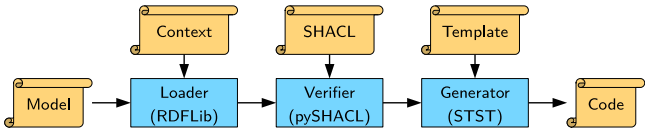

Fig. 3. Code generation workflow with models or code (orange scrolls) and tools (blue boxes).

III. Implementation

To realize the motion specification, it must be transformed and executed on the robot as we discuss in this section. Additionally, we provide background information on the solvers required to compute commands for the robot's base and arms.

A. Transforming Models to Executable Code

The models above mostly focus on the *declarative* description of a contact-rich manipulation task, that is, which constraints the robot should satisfy. Hence, the next step is to generate correct-by-construction code that can execute the task. The associated workflow is depicted in Fig. 3. We rely on the StringTemplate [14] engine and its STST[4] frontend in conjunction with custom C++ templates. StringTemplate enforces templates without logic, hence, fostering the clear separation of any model processing from the underlying template rendering.

To implement that logic we employ RDFLib[5], a Python-based triple/quad store with associated utilities for serializing and deserialising models in the Resource Description Format (RDF)[6] or JSON-LD, as well as querying and modifying the in-memory graph. Then we rely on the pySHACL[7] Python library to verify structural model constraints specified in SHACL. Using RDFLib's native API as well as its SPARQL[8] interface, we collect the various fragments required for the *imperative* algorithms that realize the robot's behaviour. On the one hand, these fragments are the *data structures* in the motion specification which comprise the coordinate representations of distances, geometric relations or forces. On the other hand, they include the *operators* such as the distance computations, the evaluation of the associated constraints or the monitors and controllers that act on the data structures. From this information we construct a dedicated JSON-based intermediate representation tailored towards the templates. The templates then transform the algorithms to executable code. First, they realize the concrete syntax for C++ as well as any required libraries. This also includes mapping the data structures to C++ variables. Then, they connect the operators to their associated data structures by calling the desired functions with their respective arguments. Next, they serialize the operators in an event loop, i.e. they determine the computations' order of execution. Finally, the templates also inject the communication with the robot into the event loop. This comprises reading the current state and sending the torque commands for both arms and the mobile base.

B. Dynamics Solver for the Arms

Our robot is equipped with two Kinova Gen 3[9] arms, each with seven degrees of freedom and torque interfaces for all its actuators. In this work, we have opted for the Kinematics and Dynamics Library (KDL)[10] to provide the recursive solvers for the arms. Specifically, we rely on the acceleration-constrained hybrid dynamics algorithm (ACHDA) [15], [16], [17]. The ACHDA can handle partial acceleration specifications, i.e. not all six degrees of freedom of a link (usually the end-effector) must be constrained. Task modelling for this solvers starts from the least-constrained motion which is the free fall under the influence of gravity. This unconstrained motion is also the "natural" redundancy resolution criterion so that there is no need for a low-priority "posture" or "auxiliary" task that is commonly found in other task specification frameworks.

C. Force Distribution Solver for the Base

The mobile base consists of four KELO Drives[11], each with two actuated wheels that are arranged like on an office chair: a differential-wheeled configuration that is connected to the platform by a pivot joint which, in turn, is displaced by a castor offset. As a result the mobile base is always singularity-free albeit with varying efficiency of force transmission from the drives to the platform. That efficiency depends on the drive's orientation with respect to the required platform-level wrench $\boldsymbol{F}_p = [f_x, f_y, m]$ with two planar forces and a moment around the vertical axis. This wrench must be distributed to the drive forces $\boldsymbol{F}_d = [f_{1,x}, f_{1,y}, \ldots, f_{4,x}, f_{4,y}]$.

We have developed the `hddc2b`[12] library and a code generator that allows us to declaratively describe a wide range of force distribution solvers. Here, we concretely rely on a solver for the weighted least squares problem in Eq. (2).

$$\begin{aligned} \min_{\boldsymbol{F}_d} \quad & \frac{1}{2}\|\boldsymbol{F}_d - \bar{\boldsymbol{F}}_d\|^2_{\boldsymbol{W}} \\ \text{subject to} \quad & \boldsymbol{G}\boldsymbol{F}_d - \boldsymbol{F}_p = \boldsymbol{0} \end{aligned} \tag{2}$$

The 3×8 force composition matrix $\boldsymbol{G}$ depends on the pivot joint angles $\boldsymbol{q}_{pvt}$. $\bar{\boldsymbol{F}}_d$ is a reference drive force and $\boldsymbol{W}$ a 8×8 positive-definite weight matrix in drive space. The well-known solution to this problem [18] is given by Eq. (3).

$$\boldsymbol{F}_d = \boldsymbol{G}^{\dagger}_{\boldsymbol{W}^{-1}}\boldsymbol{F}_p + \alpha \boldsymbol{N}_{\boldsymbol{W}^{-1}}\bar{\boldsymbol{F}}_d \tag{3}$$

Here, $\boldsymbol{N}_{\boldsymbol{W}^{-1}} = \boldsymbol{I} - \boldsymbol{G}^{\dagger}_{\boldsymbol{W}^{-1}}$ is the nullspace projector associated with the weighted pseudoinverse $\boldsymbol{G}^{\dagger}_{\boldsymbol{W}^{-1}}$. A projection is a homogeneous map which generally reduces the magnitude of its input. Hence, we introduce the scaling factor α to recover the magnitude of the greatest reference drive force.

With sufficiently high platform forces, the drives can eventually passively align towards the best force transmission efficiency. However, to reduce these platform forces it is preferable to "nudge" the drives to that alignment with the lower-priority reference force. To this end, the drive alignment is modelled as two competing tasks: one for aligning the drives with the platform's moment, the other for the alignment with the platform's

[4] https://github.com/jsnyders/STSTv4
[5] https://github.com/RDFLib/rdflib
[6] https://www.w3.org/TR/rdf11-concepts/
[7] https://github.com/RDFLib/pySHACL
[8] https://www.w3.org/TR/sparql11-query/
[9] https://www.kinovarobotics.com/product/gen3-robots
[10] https://www.orocos.org/kdl.html
[11] https://www.kelo-robotics.com/technologies/#kelo-drives
[12] https://github.com/comp-rob2b/hddc2b

TABLE I
OVERALL NUMBER OF MODEL ENTITIES/LINES

	World model	Constraints	Controllers	3D-1D Maps	Solver specification
Single Arm	35/446	7/78	12/138	7/79	10/105
Two Arms	89/1128	18/200	24/276	20/200	20/210
SC1	100/1258	18/200	34/416	20/200	24/248
SC2	100/1258	18/200	34/416	20/200	26/266

force. The alignment torque is then computed as the weighted sum of both tasks.

IV. EXPERIMENTAL EVALUATION

In this section we evaluate our approach in two steps by analysing the models we have developed and by conducting experiments on a real-world, bi-manual mobile robot.

A. Incremental Construction of Models

Evaluating (meta) models is a challenging endeavour. Hence, we provide anecdotal evidence from the implementation of the alignment behaviour. We had started with the models for a *single* arm to keep the contact with the table via a feed-forward force controller while keeping the end-effector at a stationary height. During the execution of that specification we observed that the elbow dropped under the influence of gravity which we consequently inhibited by a constraint on its height. Next, we noticed the end-effector slowly sliding along the table's edge in the lateral direction, but also in the vertical direction, while additionally changing its orientation. We stabilized these motions by constraints on the position or velocity level. The above constraints are regulated by manually-tuned PID controllers that compute a force for the elbow and acceleration constraint setpoints for the end-effector. As a next step, we introduced the *second* arm with a motion specification analogue to the first one. Finally, to realize the alignment behaviour, we constrained the shoulder-contact distance of both arms to a bilateral region, or tube, and enforced that distance by a PID-like controller with a *decaying* integral term as explained below. More details are available in the tutorial.[13]

We argue that this incremental development approach succeeded *because* of the models' composability and compositionality. The former lets us gradually add to the existing models while the latter means that (i) the robot's behaviour is predictable after introducing a change to the models; and (ii) this addition does not interfere with the already modelled behaviour. Table I shows the size of the resulting models measured in the number of model entities and the number of lines per domain.

B. Realization of Two Variants

To evaluate the robot's behaviour we have realized two variants of the alignment scenario on a real robot: (i) aligning with an actively-controlled mobile base while the arms only maintain contact with the table's edge similar to whiskers; and (ii) aligning through forces exerted by the arms to the table while the base remains passive. The majority of models and software functions is reused between both scenarios as is also evident from the accompanying repository.

In the following, for both cases, we focus on the distances between the shoulders and the contact points which the robot

[13] https://secorolab.github.io/motion-spec-ral/tutorial.html

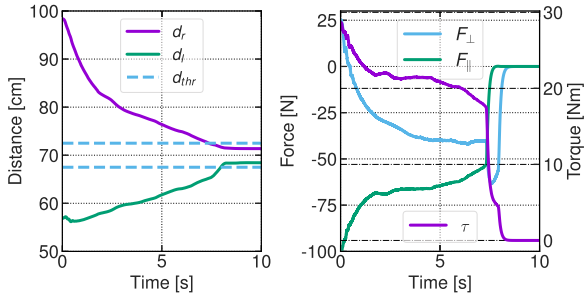

Fig. 4. Result for the base alignment in *SC1*. The left side shows the evolution of the shoulder-contact distances for both arms (d_r and d_l) in comparison to the thresholds d_{thr}. The right side depicts the control force perpendicular ($F_\perp$) and parallel ($F_\parallel$) to the table, as well as the torque τ. $F_\perp$ and $F_\parallel$ are measured on the left scale but τ on the right scale.

estimates based on the arms' forward position kinematics. The other constraints are handled in an analogue way. First, the robot determines the distance error $\epsilon(t)$ at the current time t. Then it maps this error to a force using the PID-like controller shown in Eq. (4).

$$f(t) = K_p\,\epsilon(t) + K_i \int_0^t e^{\lambda(\tau - t)}\,\epsilon(\tau)\,\mathrm{d}\tau + K_d \frac{\mathrm{d}\epsilon(t)}{\mathrm{d}t} \tag{4}$$

Here K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively. The integral component decays exponentially with a rate $0 \leq \lambda \leq 1$. $f(t)$ is the controller's scalar force output. Next, the robot evaluates the computation from Listing 5 that maps the resulting 1D control forces to 3D space using Eq. (1).

SC1 demonstrates a sideways motion. Here, the resulting 3D forces are first transformed to the mobile base's reference frame. Both resulting wrenches are added to a single wrench that is projected to the horizontal plane to command the base. Because the arms act as levers during this transformation, the base feels not only forces perpendicular ($F_\perp$) and parallel ($F_\parallel$) to the table but also a torque (τ). For the misaligned setup depicted in the left image of Fig. 1, the resulting alignment behaviour is shown in Fig. 4. Both shoulder-contact distances gradually approach the tube that is spanned by the upper and lower threshold. Once a distance enters into the tube (around 7 s), thus satisfying the motion constraints, the distance error evaluates to zero so that the controller's proportional and derivative terms stop producing command forces and torque. A "traditional" integral term would retain its value and push the arms out of the desired tube on the opposite side. Contrarily, the *decaying* integral term transitions to zero: for a brief while the controller pushes the arm further into the tube to then keep it in a steady state. The final state of the robot is shown in the right image of Fig. 1.

SC2 demonstrates a backward and pushing behaviour. The forces are transformed to the respective arms' end-effectors, causing the arms to exert a force on the table and thereby pushing or pulling the overall robot. The left image in Fig. 5 shows the initial configuration for this scenario, where the arms are holding the table's edge with a custom mount. To align, the robot must push against the table to move back, as shown in the right image. Across ten independent trials the imposed constraints are met and the same motion emerges each time, demonstrating that, once the specification is fixed, the synthesised control behaviour is deterministic. Nonetheless, as emphasised in our incremental development approach, such repeatability does not

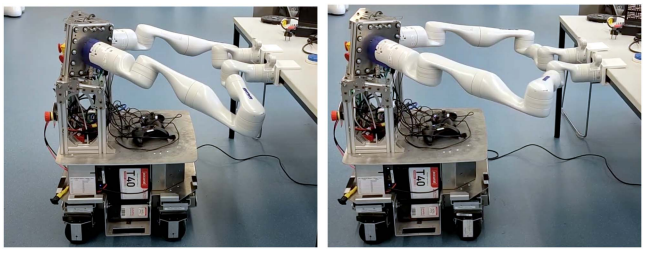

Fig. 5. Real-world experiment *SC2* showing the robot pushing the table's edge with its two arms to move backward.

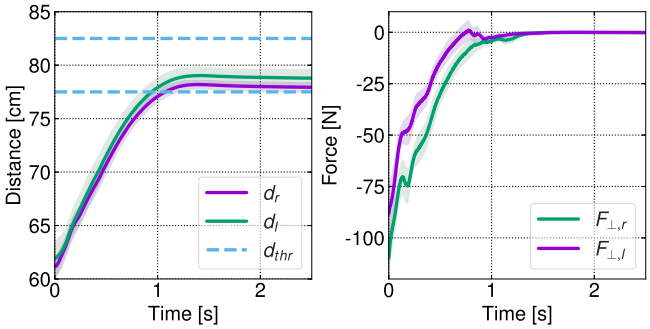

Fig. 6. Result for the base alignment in *SC2*, plotted as the mean of 10 runs (shaded regions for one standard deviation). The left side shows the shoulder-contact distances of both arms (d_l and d_r) in comparison to thresholds d_{thr}. The right side depicts the forces exerted perpendicularly to the table ($F_{\perp,l}$ and $F_{\perp,r}$). Negative forces correspond to a pushing behaviour.

guarantee that the specification of the scenario is complete. The corresponding plots in Fig. 6 display the robot's behaviour over time.

V. Discussion

We argue that, similar to humans, modern robots should be *seeking for* instead of avoiding contacts to guide their motions as we have demonstrated with the above scenarios. State-of-the-art motion planning libraries for *collision-free* motion, such as sampling-based planners like MoveIt![14] and Tesseract[15], or constraint-based task-space controllers like Giskard [19], explicitly disregard such behaviours. Indeed, a compliant interaction task such as aligning to a workspace as shown in Fig. 1 typically relies on exteroceptive information to perceive the features in the environment and to guide the robot into alignment. Here, leveraging proprioceptive information simplifies this approach while exploiting contacts reduces the need for additional sensors.

In our prior work [8] we have introduced models for kinematic chains that represent *natural* or physical constraints. This letter builds on and complements those models by *artificial* or task-level constraints. Additionally, the cross-domain findings of anti patterns in both, kinematic chains and motion specifications, supports our chosen analysis methodology.

We consider the models as "semantic interfaces" between different tools and even robots. The models not only facilitate constraint verification on data structures, but also on operators and, hence, compare favourably with APIs in general-purpose computer programming language that often only allow type checking on function arguments. Additionally, such models are a step towards a vendor-independent motion specification where the same description produces consistent behaviour across different robots. Furthermore, the explicit models enable a robot to check in advance if it can execute such a specification, meaning if it can understand all model entities. Beside the above static code generation, a robot could also be equipped with a runtime interpreter for such models.

JSON-LD lays the foundation for our work by providing the primitives to structure the models, but the composable and compositional metamodels must be correctly designed by experts using their knowledge of a domain. In particular, a future composition over the models presented here are trajectory generators that adapt the controllers' setpoints or thresholds over time. Such a composition is straightforward to realize *because* these quantities are reified in our models. A further point of integration, that we have explicitly excluded from our models, is the time-discrete coordination or scheduling of multiple motion specifications. Hence, state machines as used in FlexBE xDL [20], Petri nets [13] or task-planners akin to the MoveIt! Task Constructor [21] can refer to and, thus, compose these motion specifications.

By design, we have opted for an *ego-centric* perspective for the models, meaning that the constraints are all specified with respect to the robot itself. However, the same concepts are also required for an *allo-centric* task specification that relates the robot to its environment. For instance, an additional constraint could be that the robot should not move the table while performing its alignment task. This requires more complete world models with associated estimators.

In a similar spirit, solving *design problems* remains beyond the scope of our toolchain which does not validate that the robot's resulting behaviour correctly solves a specific task. Such design problems comprise checking if all motion constraints are specified or that the chosen controllers and their parameters are good enough for the task at hand. What we do achieve is correct-by-construction code via two factors. First, the structural model constraints via SHACL allow the verification of the models' well-formedness. Second, once the templates have been validated, they can generate correct code for any number of those models.

The incremental development paradigm for functionally correct *software* originates from Dijkstra's work [22] that allows developers to refine applications using sound relations derived from first principles. The traditional correctness-by-construction framework uses a Hoare triple that consists of a precondition, an abstract statement, and a postcondition. Refinement rules ensure the triple's validity, leading to a concrete implementation where each rule's soundness is proven to maintain correctness. The statements in Dijkstra's work represent the *imperative* instruction set of an abstract computer. In contrast, we *declaratively* describe the motions of a robot; the computer representation enters afterwards.

VI. Conclusion

To investigate how robots can simplify task execution by using environmental contacts as guides, we propose defining *composable* and *compositional* models that can be used to build motion specifications as a set of constraints on the geometric relations and the forces among the features of the robot and

[14] https://moveit.ai/
[15] https://github.com/tesseract-robotics/tesseract

the environment. These constraints can either be controlled or monitored to achieve a desired motion. Through such models, we exploit the proprioceptive information that is inherent to the robot such as kinematics (joint positions and velocities) and dynamics (forces and torques) and impose constraints on them which nurtures the robot's motion to adapt naturally. We have implemented our approach on a real-world, dual-arm redundant mobile manipulator interacting with a workspace by utilizing contacts to guide its alignment.

As future work we foresee the development of complete (meta) models of the computational architecture that are currently only implicitly encoded in the templates. Additionally, we are planning to bring all the models to the robot's runtime so that the robot can adjust its behaviour online.

REFERENCES

[1] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks," in *Proc. Int. Conf. Adv. Robot.*, 2009, pp. 1–6.

[2] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," *Int. J. Humanoid Robot.*, vol. 2, no. 4, pp. 505–518, 2005.

[3] C. -L. Fok, G. Johnson, L. Sentis, A. Mok, and J. D. Yamokoski, "ControlIt! — A software framework for whole-body operational space control," *Int. J. Humanoid Robot.*, vol. 13, no. 1, 2016, Art. no. 1550040.

[4] J. D. Schutter et al., "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Int. J. Robot. Res.*, vol. 26, no. 5, pp. 433–455, 2007.

[5] W. Decré, R. Smits, H. Bruyninckx, and J. D. Schutter, "Extending iTaSC to support inequality constraints and non-instantaneous task specification," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 964–971.

[6] G. Borghesan, E. Scioni, A. Kheddar, and H. Bruyninckx, "Introducing geometric constraint expressions into robot constrained motion specification and control," *IEEE Robot. Automat. Lett.*, vol. 1, no. 1, pp. 1140–1147, Jul. 2016.

[7] E. Aertbeliën and J. D. Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 1540-1546.

[8] S. Schneider, N. Hochgeschwender, and H. Bruyninckx, "Domain-specific languages for kinematic chains and their solver algorithms: Lessons learned for composable models," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 9104-9110.

[9] T. De Laet, S. Bellens, R. Smits, E. Aertbelien, H. Bruyninckx, and J. D. Schutter, "Geometric relations between rigid bodies (Part1): Semantics for standardization," *IEEE Robot. Autom. Mag.*, vol. 20, no. 1, pp. 84–93, Mar. 2013.

[10] S. Schneider, N. Hochgeschwender, and H. Bruyninckx, "Semantic composition of robotic solver algorithms on graph structures," *Front. Robot. AI*, vol. 29, no. 11, 2024, Art. no. 1363150.

[11] J. W. Hill and A. J. Sword, "Manipulation based on sensor-directed control: An integrated end effector and touch sensing system," in *Proc. Annu. Human Factors Soc. Conv.*, NASA Rep. NASA-CR-143420, 1973.

[12] M. T. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Trans. Syst., Man, Cybern.*, vol. TSMC-11, no. 6, pp. 418–432, Jun. 1981.

[13] M. I. Artigas, R. T. Rodrigues, L. Vanderseypen, and H. Bruyninckx, "Software patterns and data structures for the runtime coordination of robots, with a focus on real-time execution performance," *Front. Robot. AI*, vol. 4, no. 11, Sep. 2024, Art. no. 1363041.

[14] T. J. Parr, "Enforcing strict model-view separation in template engines," in *Proc. Int. Conf. World Wide Web*, 2004, pp. 224–233.

[15] E. P. Popov, A. F. Vereshchagin, and S. L. Zenkevich, *Manipuljacionnyje Roboty: Dinamika I Algoritmy*, D. S. Furmanov, Ed. Moscow, Russia: Moscow Nauka, 1978.

[16] A. F. Vereshchagin, "Modeling and control of motion of manipulational robots," *Sov. J. Comput. Syst. Sci.*, vol. 1, pp. 125–134, 1989.

[17] S. Schneider and H. Bruyninckx, "Exploiting linearity in dynamics solvers for the design of composable robotic manipulation architectures," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7439–7446.

[18] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses: Theory and Applications*, 2nd ed., J. Borwein and P. Borwein, Eds. Berlin, Germany: Springer, 2003.

[19] S. Stelter, G. Bartels, and M. Beetz, "An open-source motion planning framework for mobile manipulators using constraint-based task space control with linear MPC," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 1671–1678.

[20] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 2796–2802.

[21] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "MoveIt! Task constructor for task-level motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 190–196.

[22] E. W. Dijkstra, *A Discipline of Programming*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1976.