Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-26)
Special Track on AI and Robotics

Closing the Motion Execution Gap: From Semantic Motion Task Constraints to Kinematic Control

Simon Stelter , **Vanessa Hassouna** , **Malte Huerkamp** and **Michael Beetz**
University of Bremen
{stelter, hassouna, huerkamp, mbeetz}@uni-bremen.de

Abstract

This paper addresses the Motion Execution Gap, the disconnect between high-level symbolic task descriptions using semantic constraints and executable robot motions. **Motion Statecharts** are introduced as an executable symbolic representation for complex motions. They allow the arbitrary arrangement of motion constraints, monitors or nested statecharts in parallel and sequence. World-centric motion specification and generalization across embodiments are enabled through the use of a unified differentiable kinematic world model of both, robots and environments. Motion execution is realized through a linear Model Predictive Control (lMPC)-based implementation of the task-function approach, in which smooth transitions during task switches are ensured using jerk bounds. Cross-platform transferability was demonstrated by deploying the method on eight robot platforms, operating in diverse environments. The proposed framework is called Giskard and is available open source (https://github.com/cram2/cognitive_robot_abstract_machine).

1 Introduction

When describing robot motions, semantic constraints provide a natural and expressive approach. Consider "make a salad", as shown in Fig. 1 where a PR2 robot prepares ingredients by cutting a zucchini. Such instructions are represented in cognitive robotics as symbolic tasks specifying **what** to achieve, not **how** to move. High-level actions are decomposed into sub-tasks (cutting, grasping, placing) using Generative AI [Gupta *et al.*, 2024] or plan executives [Beetz *et al.*, 2010; Fox and Long, 2003], often leveraging knowledge graphs [Kümpel *et al.*, 2025]. For example, "cut the zucchini into small slices" yields explicit constraints (slice thickness) and implicit ones (safe knife handling near humans), separating valid from invalid trajectories. A system remains essential to convert these symbolic, constraint-rich descriptions into continuous motion satisfying all constraints.

Complete cognitive robot architectures such as SkiROS2 [Mayr *et al.*, 2023], ArmarX [Wächter *et al.*, 2016], and CRAM [Beetz *et al.*, 2023] are designed to achieve transferability across robots by hierarchically resolving symbolic motion tasks down to primitive Cartesian pose or joint-space goals. Those primitive poses are typically further divided into mobile base and manipulator tasks [Sandakalum and Ang Jr, 2022] and they assume that tasks can be grounded at planning time [Pan *et al.*, 2024]. This design inherently limits the class of symbolic motion tasks that can be defined. A task such as "open the fridge" cannot easily be reduced to these primitives, as it requires a discrete arch trajectory for the hand and the robot's mobile base to be positioned appropriately [Bozcuoğlu *et al.*, 2018]. Motions requiring degrees of freedom, like pouring from a jug at an arbitrary angle, are more naturally expressed with geometric constraints [Borghesan *et al.*, 2015]. This limitation constitutes the "Motion Execution Gap".

Figure 1: Illustrating the Motion Execution Gap: PR2 must generate constraint-satisfying trajectories from semantic instruction "cut zucchini into small slices".

Addressing the motion execution gap through generative AI remains infeasible. Due to their probabilistic nature, such methods cannot provide formal guarantees for constraint adherence [Huang *et al.*, 2025]. Even state-of-the-art Vision-Language-Action models (VLAs) require embodiment-specific post-training for transferability across robotic platforms [Kim *et al.*, 2025].

A popular alternative is to formulate motions as constraint optimization problems. The task function approach expressed as a Quadratic Program (QP), often termed QP-control [Aertbeliën and De Schutter, 2014; Bouyarmane *et al.*, 2018;

Stelter *et al.*, 2022; Escande *et al.*, 2014], is a promising variation. Several works integrate QP-based task-function control with symbolic motion representations: In [Pane *et al.*, 2021], a QP-controller pairs with a finite state machine (FSM) representation, executing motion tasks in parallel while FSM transitions serve as monitors. [Bolotnikova *et al.*, 2021] similarly combine FSMs with QP-control for sequential/parallel multi-limb tasks. [Domínguez *et al.*, 2022] attaches a low-frequency Behavior Tree (BT) to a high-frequency QP-controller, dynamically adding/removing tasks via implicit end conditions for smoothness. [Haviland *et al.*, 2022] employs BTs for pick-and-place sequencing, focusing on discrete motions rather than continuous task composition. [Rovida *et al.*, 2018] extends BTs with parallel composites for concurrent impedance-controlled "skills", merging commands from parallel nodes. FSMs and BTs are common choices, though FSMs suffer state explosion and BTs face computational scaling issues.

Motion Statecharts (MSCs) are introduced as statecharts specialized for motion composition. Statecharts, first introduced by [Harel, 1987], have become a general term for extensions of FSMs aimed at alleviating the state explosion problem for specific use cases. MSCs enable symbolic reasoning over motion tasks and support online monitoring and adaptation akin to hybrid symbolic/neural planning.

However, task transitions commonly introduce discontinuities, typically mitigated by slowing down during transition [Domínguez *et al.*, 2022] or gradual task weight scaling [Pane *et al.*, 2021], both limiting the motion representation. For instance, single-task switches necessitate system-wide deceleration, impacting parallel tasks. The proposed system instead formulates task functions as short-horizon lMPC with explicit jerk limits, ensuring a smooth velocity motion profile. MSCs are thus allowed to swap motion tasks at any time without concern for the underlying controller.

In most of the aforementioned task function-based systems, only the robot kinematics are modeled. In this paper, it is instead proposed to model the kinematics of the environment as well to enable motions to be defined in a world-centric manner. This allows the definition of the previous "open the fridge" example with two constraints, hold the handle and move the door according to its articulation model. In [Bouyarmane *et al.*, 2018], a similar idea is explored, although the approach is based entirely on URDF [ROS.org, 2022], which is limited in its descriptive capabilities. The approach proposed in this paper is instead similar to [Röfer *et al.*, 2022], by which even complex dependent kinematics can be described. Both are built on top of CasADi [Andersson *et al.*, 2019], a library for computer algebra and automatic differentiation designed specifically for optimal control.

The proposed framework is called **Giskard** and uniquely combines the following components to close the motion execution gap.

- **Motion Statecharts** are provided as executable symbolic motion representations, by which online monitoring and automatic semantic annotation of trajectories are enabled.
- A **differentiable kinematic world model** is provided as a structured semantic digital twin, by which world-centric task functions are supported.
- **lMPC-based task-function control** combines active constraints dictated by a MSC with explicit jerk bounds to prevent discontinuities during task transition.

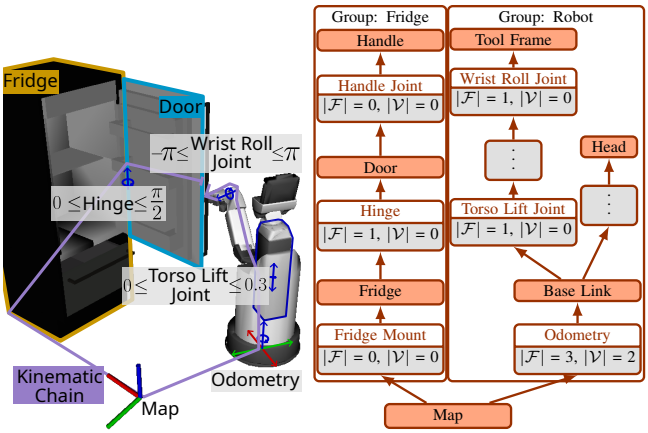

Figure 2: An example of a world containing a fridge and a Toyota HSR (left), with its kinematic model (right).

Together these components not only allow motion descriptions via semantic constraints, but also offer real-time semantic feedback during the motion. This lets the high-level planner implement a closed semantic feedback loop with the motion executive. It can understand why motions succeed or fail, and adjust plans, instead of treating execution as a black box.

Transferability was validated on eight diverse platforms, as discussed in Chapter 5.2, ranging from mobile manipulators such as the PR2 to static dual-arm setups. Minimal robot-specific tuning was required, by which out-of-distribution generalization was demonstrated.

2 Differentiable Kinematic World Model

Giskard requires a kinematic model to describe motions with task functions. An example world model is shown in Fig. 2. The model is defined as follows.

- $\mathcal{L}$: A set of links.
- $\mathcal{J}$: A set of joints.
- $\mathcal{S} = \mathcal{F} \cup \mathcal{V}$:
  - $\mathcal{F}$: Symbols referring to Degrees of Freedom (DoFs). This set includes robot's DoFs, e.g., "wrist roll joint", as well as environmental DoFs, e.g., "hinge". For each such symbol q, its time derivatives up to the third order, namely $\dot{q}, \ddot{q}$, and $\dddot{q}$ are also included in $\mathcal{F}$.
  - $\mathcal{V}$: Symbols representing non-actuated variables, treated as constant by the controller, e.g., robot localization.
- $\mathcal{C}$: A set of constraints limiting the range of symbols in $\mathcal{F}$.

Links and joints form a tree structure that includes both the robot and the articulated environment. Joint transformations are defined as functions over $\mathcal{S}$.

$$^{\texttt{parent}}T_{\texttt{child}} = f(\mathcal{S}) \in \mathrm{SE}(3) \tag{1}$$

They are implemented as symbolic expression using CasADi [Andersson *et al.*, 2019] for automatic differentiation. A distinction is made between joints and DoFs, since a

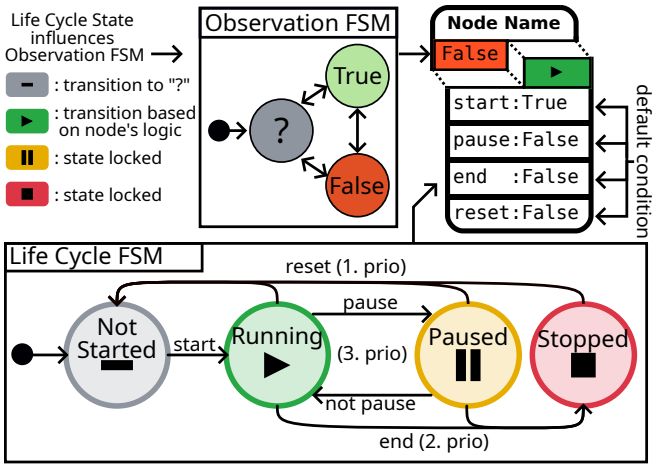

Figure 3: Leaf MSC node with life cycle FSM (bottom) and observation FSM (top left): `True`/`False`/`"?"` tracks task achievement for transitions *and* trajectory annotation.

many-to-many relationship exists between them. For example, the "Odometry" joint contains non-actuated variables to encode positional measurements relative to "map" and three DoFs to encode the robot's ability to translate and rotate relative to "Base Link". DoFs of the articulated environment are treated as controllable, for example in the case of a "Hinge", since the robot can exert indirect control over them while in contact. This approach simplifies the definition of environment manipulation tasks, as discussed in the next subsection.

2.1 Differentiable Task Functions

The main purpose of the world model is to provide a means of computing forward kinematics (FK) for chains between any link pairs. FKs are computed by multiplying the transformation matrices of all joints along the chain. Fig. 2 shows a handle-to-gripper chain spanning both the robot and the environment. All FKs are automatically differentiable thanks to CasADi and can be used in the definition of task functions.

The high-level idea of the task function approach is to describe a motion as a set of task functions. Task functions are constraints defined on differentiable task spaces. A **task space** (f) maps the world state onto an arbitrary space:

$$f : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{m \times n},$$

where m and n are arbitrary integers. The robot must be capable of influencing the task space. Technically, this means that at least one partial derivative with respect to the robot's DoFs is non-zero. FKs that include parts of the robot satisfy this definition. In Fig. 2, this enables the description of a door-opening task by combining a joint-space task for the "Hinge" with a Cartesian-space task for the gripper to hold the handle. Furthermore, these expressions can be extended with arithmetic and trigonometric operations provided by CasADi to produce arbitrary geometric constructs.

Motions are described by constraining task spaces using task functions. A **Task Function** is an (in-)equality constraint on a scalar-valued task space: $f_\epsilon = b_\epsilon$, or $l_\xi \leq f_\xi \leq u_\xi$. For example, a 6D Cartesian-space motion goal is defined using six equality constraints, three for position and three for rotation, with f_ϵ derived from the FK and b_ϵ being the position/rotation error. Grasping a handle can be implemented with a task space describing the distance between the robot's gripper and a line representing the handle. A task function can constrain that distance to be equal to zero. The resulting motion moves the gripper to its closest point on the handle.

3 Motion Statecharts

Consider a high-level planner that provides a symbolic action description such as "open the fridge". It may produce a sequence such as "grasp the handle" $\rightarrow$ "move the door open" $\rightarrow$ "release the handle", or more granular steps. These actions must be grounded into continuous control commands. Furthermore, to execute this sequence as a single motion, a single set of task functions is insufficient because the constraints for opening the door must not be active during the handle-grasping phase.

Previous work proposes the use of hierarchical FSMs [Pane *et al.*, 2021; Bolotnikova *et al.*, 2021] or BTs [Domínguez *et al.*, 2022; Haviland *et al.*, 2022; Rovida *et al.*, 2018] to group task functions into different phases. The structure of BTs makes them difficult to integrate directly into the control loop, as all three cited works run them at lower frequency (2–10 Hz) in parallel with high-frequency controllers. FSMs do not share this limitation and support real-time execution, but suffer from the state explosion problem. As system complexity increases, the number of states grows combinatorially, though hierarchical definitions mitigate this to some extent.

The proposed MSCs build on statecharts to retain FSM real-time capability while alleviating state explosion by organizing motion components into a network of dual-FSMs. This yields a reusable representation that bridges the motion execution gap by composing motion tasks and monitors into reactive graphs portable across robots and environments.

3.1 Structure and Elements

MSCs are defined as a variation of statecharts with explicit motion-related constructs tailored to robot control. They are represented as graphs whose nodes can contain nested statecharts, by which reusability and structural decomposition are promoted. All nodes may, but do not have to, be associated with motion task functions (see Sec. 2.1), which remain active for the duration of the node's activation. Nodes that are associated with task functions are referred to as **Motion Tasks**, those without are called **Monitors**. An example node is depicted in Fig. 3. Each node contains two internal FSMs.

- An observation FSM (Fig. 3, top-center) is used to represent whether an observed condition is currently `True`, `False`, or `"?"`. For Motion Tasks, this state represents the constraint satisfaction status. An explicit `"?"` state is required for situations in which the state is unknown, for example when the node has never been active.
- A life cycle FSM (Fig. 3, bottom) is used to determine whether the node is active.

Conditions are depicted in the lower half of a node (see Fig. 3, top right). They govern transitions between life cycle states,

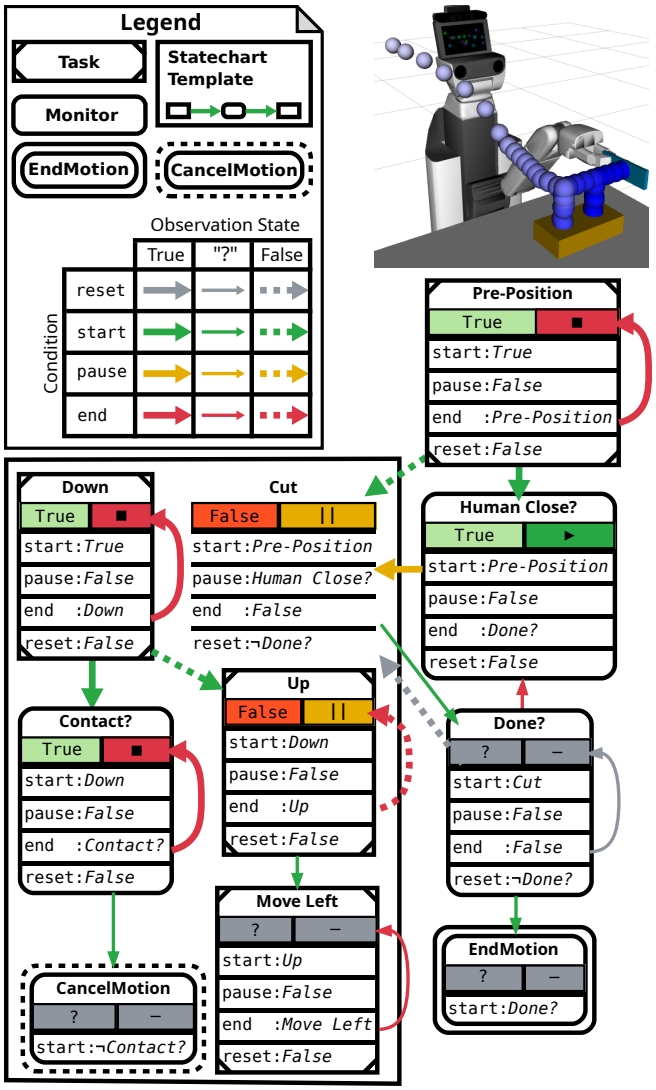

Figure 4: An example MSC for executing a cutting motion, showcasing all its features. It shows a snapshot when a "Human Close?" was observed, pausing the "Cut" MSC template.

for example transitioning a node from ▬ to ▶ when the start condition is satisfied. These transitions are expressed as ternary logical expressions that combine the three-valued observation states of any node on the same hierarchy level. In addition, all nodes may implement callbacks that are triggered during every control cycle in which the node is ▶, or when a transition between life cycle states occurs. A nested statechart is referred to as a **MSC Template**. MSC Templates are themselves nodes with the same internal FSMs structure and contain isolated statecharts. The life cycle states of nested MSC nodes are linked to the life cycle state of the parent. If the parent node is in ▶ state, the child nodes can change state freely. In all other cases the child nodes mirror the parent's state, if the diagram in Fig. 3 allows the transition. This means a parent cannot force a child transition from ▬ to ⏸, but it can make a child transition from ▶ to ■.

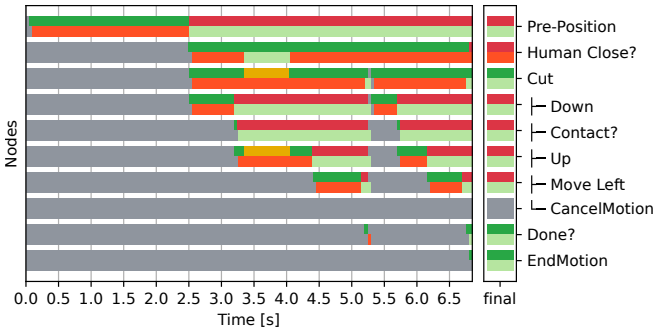

Figure 5: The semantically annotated trajectory corresponding to Fig. 4 as a gantt chart. The top bar shows life cycle and bottom bar shows observation states. The colors correspond to Fig 3.

In addition to enabling reuse of frequently occurring MSC structures, templates are used to impose structure on motions. Common examples include the following. `Sequential`: Child nodes are activated upon the predecessor reaching the `True` state, and the template succeeds when the final child succeeds. `Parallel`: All child nodes are activated simultaneously, and the template succeeds when n children reach the `True` state.

The terminal states of MSCs are referred to as **EndMotion** and **CancelMotion**. Both states trigger termination of the control loop and stop motion execution. The distinction between the two states is purely semantic feedback.

3.2 Motion Statecharts in the Control Loop

At runtime, the state of the MSC is evaluated at every control cycle in the following order:

1. The `on_tick` callbacks of all ▶ nodes are executed given the current system state, including joint states, transforms, and sensor inputs, in order to update their observation states.
2. All life cycle state machines are updated simultaneously based on the current observation states of all nodes.
3. If a final state exhibits a `True` observation state, execution terminates and zero commands are sent to the robot.
4. Otherwise, the task functions associated with all ▶ nodes are collected and applied as constraints to the lMPC.
5. The lMPC computes the next jerk commands for all DoFs of the system, which are integrated to velocity commands and sent to the robot.

3.3 Example: MSC of a Cutting Motion

Figure 4 illustrates an MSC executing a cutting motion on the HSR platform. The motion starts with the "Pre-Position" task, the only node whose start condition is `True`. The "Cut" MSC template then coordinates downward, upward, and leftward knife motions through three self-terminating tasks. The embedded monitor "Contact?" aborts the sequence if it fails to activate following "Down" completion. Upon termination of "Move Left", the template advances to a `True` observation state. The "Done?" monitor subsequently determines whether to reset the "Cut" template or end execution.

In parallel, the "Human Close?" monitor remains active, pausing the "Cut" template on human proximity detection, as depicted in the figure snapshot. At this time, "Down" and "Contact?" have already reached ■. These nodes remain in ■ because no transition to ⏸ exists (cf. Fig. 3). "Up" transitions from ▶ to ⏸, because such a transition is possible. Unstarted child nodes remain in ▬.

Node observation states serve dual purposes: triggering life cycle transitions and enabling *real-time motion annotation*. Figure 5 presents a Gantt chart of the cutting motion execution, with each node depicted by dual bars representing life cycle and observation states, respectively. Human proximity is evident between $t \approx 3.4$ and $t \approx 4.1$ via the monitor's `True` observation state.

4 lMPC-Based Implementation of the Task Function Approach

MSCs allow task functions to be exchanged at every control cycle. In standard QP-controllers, this can introduce discontinuities. Alternatives defer switching until motion phase end [Domínguez *et al.*, 2022] or use weight scaling [Pane *et al.*, 2021]. The former limits motions to separable phases with pauses, the latter is unable to deactivate constraints immediately and adds tuning parameters for the weight-scaling strategies.

To address this, a lMPC-based task function implementation was developed that prevents discontinuities in the joint velocity commands using jerk limits. It predicts system evolution over a short horizon to enforce jerk limits, with task functions as constraints.

4.1 Formulation as a Quadratic Program

The lMPC is formulated as a QP:

$$\min_{\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}},\boldsymbol{\epsilon},\boldsymbol{\xi}} \quad \sum_{k=0}^{N-3} (\|\dot{\boldsymbol{q}}_k\|^2_{\boldsymbol{w}_{\dot{q},k}}) + \|\boldsymbol{\epsilon}\|^2_{\boldsymbol{w}_\epsilon} + \|\boldsymbol{\xi}\|^2_{\boldsymbol{w}_\xi} \tag{2}$$

$$\text{s.t.} \quad \begin{matrix} \boldsymbol{l}_{\dot{q},k} \le \dot{\boldsymbol{q}}_k \le \boldsymbol{u}_{\dot{q},k} & \forall k = 0,\dots,N-3 \\ \ddot{\boldsymbol{q}}_{\min} \le \ddot{\boldsymbol{q}}_k \le \ddot{\boldsymbol{q}}_{\max} & \forall k = 0,\dots,N-1 \end{matrix} \quad \text{(bounds)} \tag{3}$$

$$\left.\begin{aligned} \dot{\boldsymbol{q}}_0 &= \dot{\boldsymbol{q}}_{\text{curr}} + \ddot{\boldsymbol{q}}_{\text{curr}}\Delta t + \ddot{\boldsymbol{q}}_0\Delta t^2 \\ \dot{\boldsymbol{q}}_1 &= 2\dot{\boldsymbol{q}}_0 - \dot{\boldsymbol{q}}_{\text{curr}} + \ddot{\boldsymbol{q}}_1\Delta t^2 \\ \dot{\boldsymbol{q}}_k &= 2\dot{\boldsymbol{q}}_{k-1} - \dot{\boldsymbol{q}}_{k-2} + \ddot{\boldsymbol{q}}_k\Delta t^2 \\ 0 &= 2\dot{\boldsymbol{q}}_{N-3} - \dot{\boldsymbol{q}}_{N-4} + \ddot{\boldsymbol{q}}_{N-2}\Delta t^2 \\ 0 &= -\dot{\boldsymbol{q}}_{N-3} + \ddot{\boldsymbol{q}}_{N-1}\Delta t^2 \end{aligned}\right\} \begin{matrix} \text{(system model)} \\ \forall k = 1,\dots,N-3 \end{matrix} \tag{4}$$

$$\boldsymbol{b}_\epsilon = \Delta t \sum_{k=0}^{N-3} \boldsymbol{J}_{\boldsymbol{f}_\epsilon}\dot{\boldsymbol{q}}_k + \Delta t\boldsymbol{\epsilon} \qquad \text{(eq. task func.)} \tag{5}$$

$$\boldsymbol{l}_\xi \le \Delta t \sum_{k=0}^{N-3} \boldsymbol{J}_{\boldsymbol{f}_\xi}\dot{\boldsymbol{q}}_k + \Delta t\boldsymbol{\xi} \le \boldsymbol{u}_\xi \quad \text{(neq. task func.)} \tag{6}$$

The first term of the objective function minimizes the joint velocities $\dot{\boldsymbol{q}}_k$, weighted by $\boldsymbol{w}_{\dot{q}}$, over a prediction horizon N. They are represented as vectors containing one variable per DoF of the system. N is chosen to be sufficient to enforce jerk limits, with $N\Delta t$ typically below ≤ 0.25s. The weights start at 0.001 and increase linearly to 0.01 at the end of the horizon, thereby encouraging the use of immediate velocities.

The remaining objective function minimizes vectors of slack variables $\boldsymbol{\epsilon}$ and $\boldsymbol{\xi}$, associated with equality and inequality constraints, respectively. They prevent infeasibility in the presence of conflicting constraints through relaxation. By default, their weight is set to 1, normalized by the square of the expected maximum velocity of the corresponding task. As a consequence, using joint velocity to fulfill constraints is approximately two orders of magnitude cheaper than violating them using slack variables.

The jerk decision variables $\ddot{\boldsymbol{q}}$ of the objective function influence it only indirectly. Instead, the **system model** links velocity and jerk through constraints derived from a semi-implicit Euler integration scheme,

$$\dot{\boldsymbol{q}}_k = \dot{\boldsymbol{q}}_{k-1} + \ddot{\boldsymbol{q}}_k\Delta t,$$
$$\ddot{\boldsymbol{q}}_k = \ddot{\boldsymbol{q}}_{k-1} + \dddot{\boldsymbol{q}}_k\Delta t.$$

Consequently, the weights applied to velocity variables implicitly influence the jerk variables. The system model in Eq. 4 is obtained by eliminating acceleration variables, enforcing a terminal velocity of zero for stability, and linking the current states to the initial system state. Explicit acceleration variables are omitted because their bounds had negligible impact in practice. This reduces the number of decision variables by roughly $\frac{1}{3}$. Introducing explicit jerk penalties to the objective function can destabilize the system by making continued motion cheaper than deceleration.

The **bounds** defined in Eq. 3 impose direct limits on velocities and jerks. Given the prediction horizon and desired velocity limits, a theoretical maximum jerk exists that allows the system to decelerate from the velocity limit within the horizon. This value is computed once by solving a modified version of Eq. 2 without jerk bounds. Empirically, no benefit was observed from further manual tuning of jerk limits. As a result, N and Δt are the only parameters requiring robot-specific tuning. Joint position limits are encoded into the velocity bounds by progressively restricting allowed values as joints approach their limits.

Finally, **task functions** are incorporated into the final two constraint sets in Eq. 5 and 6 by constraining the Jacobian integral over the horizon and relaxing it with slack variables. In Eq. 5 $\boldsymbol{b}_\epsilon$ corresponds to the task space error. In Eq. 6 $\boldsymbol{l}_\xi$ and $\boldsymbol{u}_\xi$ define lower and upper bounds on the task space error, respectively. To prevent excessively large error terms, all three bounds are clamped to values achievable within the horizon. The slack variables $\boldsymbol{\epsilon}$ and $\boldsymbol{\xi}$ relax these constraints as described previously. They represent the residual task space velocity integral over the horizon and are scaled by Δt such that their magnitudes are comparable to the integral term. This formulation requires only a single constraint and a single slack variable per task function in the QP. The majority of motions encountered in mobile manipulation can be expressed using such positional task functions, which supports scalability of the overall system. Velocity task constraints can also be formulated, but they require one constraint per discretization step of the prediction horizon. Further details on this formulation are provided in [Stelter, 2025].

For this application, the fastest available QP solver is qpSWIFT [Pandala *et al.*, 2019], which enables execution of the complete control loop at 100 hz on a high-performance CPU.

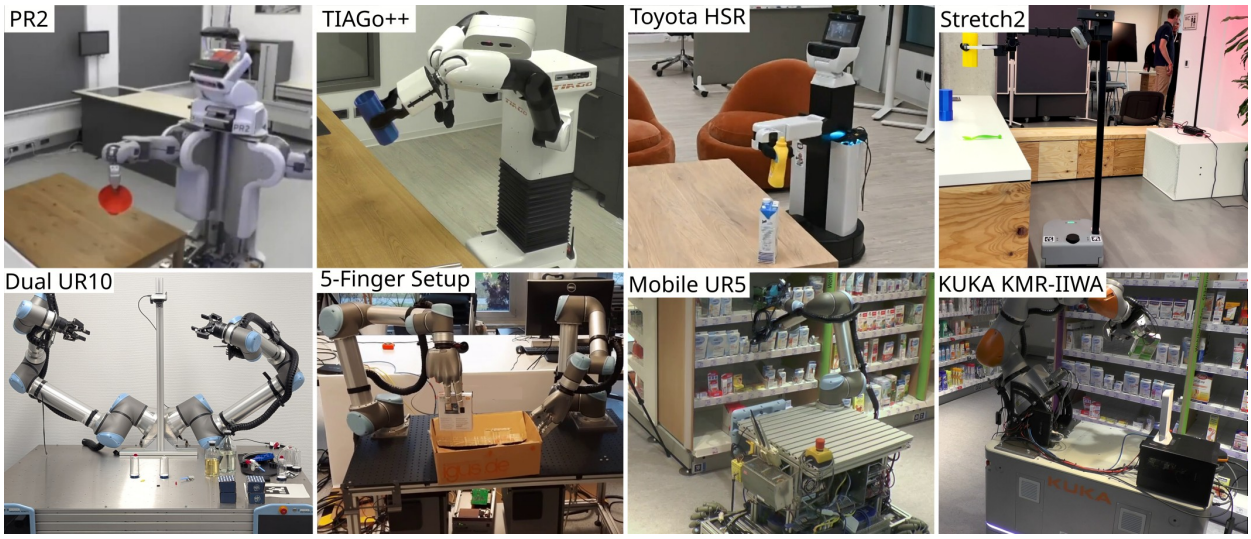

Figure 6: The eight real robots the proposed framework has been deployed on.

5 Evaluation

The presented motion executive aims to close the motion execution gap, by offering a method to describe motions semantically with MSC and translating it into a kinematic lMPC for a given robot. The framework was deployed on eight robot platforms and several different environments, shown in Fig. 6.

5.1 Closing the Motion Execution Gap with MSC

To test if the presented framework is able to close the motion execution gap, it was integrated as the motion executive into the CRAM cognitive architecture [Beetz *et al.*, 2023]. This architecture uses a knowledge base to specialize its plans and parameterize motion designators, which serve as the symbolic counterpart to MSCs in our framework. To integrate our approach, the high-level planner was modified to resolve plans to MSCs instead.

Returning to the cutting example (Fig. 1), a task such as "cut the zucchini" does not yet specify how the cutting should be carried out. Queries to a knowledge base provide answers, for instance that the zucchini should be sliced instead of cubed, or what the preceding and following motions are. The answers allow the decomposition of the plan into sub-actions and finally MSCs, linking them to their semantic meaning. This setup allows the high-level planner to reason about the current system state during execution based on the observation and life cycle states of nodes. After the execution, it can use the final MSC state to reason about failure causes.

The high-level planner decides the next step based on this semantic feedback. It may stop the plan, repeat a failed cut, switch tools, or change settings like the target position or direction of movement. The cutting process is described by a discrete separation state $S(t) \in \{\texttt{False}, \texttt{True}\}$, where $S(t) = \texttt{True}$ means that material separation has taken place at time t. A MSC monitor tracks this semantic state using geometric and contact observations. Another monitor tracks whether the contact force at the tool frame is increasing: $C(t) = \dot{\mathbf{f}}(t) > 0$. This feedback is interpreted on the high-level by evaluated logical conditions. For example "CutInProgress" is defined as both monitors having a `True` observation state: $C(t) \wedge S(t) \Rightarrow$ CutInProgress. Under this model, cutting inside the high-level planner is no longer described using a specific method or perception routines. Instead the high-level decisions are made based on semantic feedback. This clear mapping of physical signals to semantic states lets us reason, for example, that "the knife did not touch the object" if no contact is detected.

5.2 Transferability

MSC Node	Real Robots	Kin. Sim. Tests
Cartesian pose	8/8	8/8
Joint goal	8/8	8/8
Feature Functions	8/8	8/8
Door/Drawer opening	PR2, TIAGo, HSR	6/8 (mobile only)
Cutting	PR2	8/8
Diff-drive nav	TIAGo, Stretch2	TIAGo, Stretch2
Peg-In-Hole	Dual UR10	8/8
5-finger grasp	5-Finger Setup	5-Finger Setup

Table 1: List of all implemented MSC nodes and the real robots they have been used on. Furthermore, all combinations that should be possible based on a robot's kinematics were verified in simulation.

To showcase the transferability of the framework, Table 1 list all implemented MSC nodes and the real robots they have been deployed on. Since our approach uses a kinematic world model, all implementations are, in principle, robot-agnostic and can be reused on any platform that offers the required capabilities (e.g., a differential drive for the "Diff-drive navigation" node). To verify this, the world model presented in Section 2 was used to kinematically simulate sensible combinations.

"Feature Functions" are defined as algebraic expressions that combine geometric primitives, such as points, lines, and

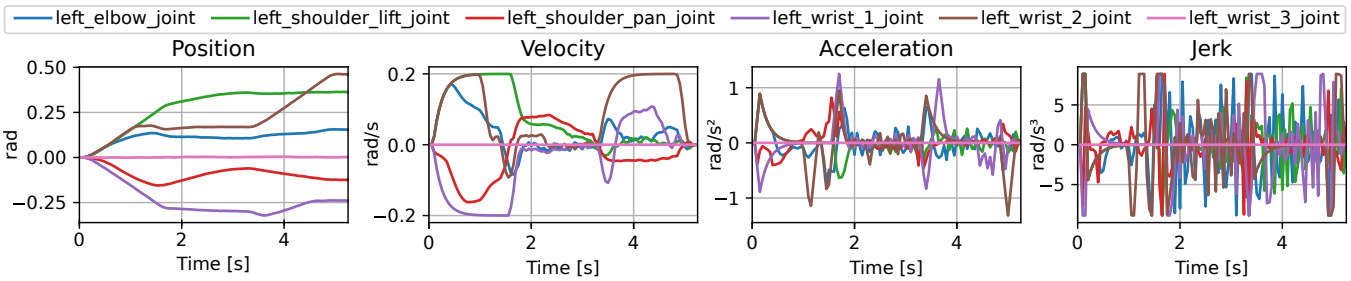

Figure 7: Joint-space trajectory of the insertion motion corresponding to Fig. 8, recorded during real-robot execution. Joint position values are offset to originate at 0 for readability.

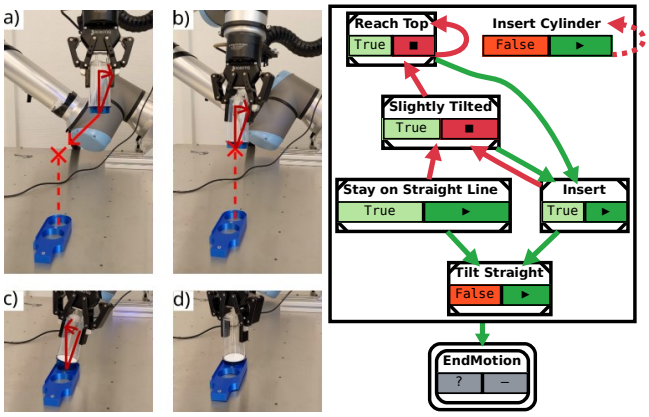

Figure 8: The dual UR10 setup executing an insertion motion and the corresponding MSC at the time of image c). Life cycle transitions of nodes are omitted to conserve space.

planes, between tool and object frames to form differentiable scalar constraints [Borghesan *et al.*, 2015]. They enable the specification of complex motions through composition within MSC. "Diff-drive navigation" addresses the limitation of platforms such as TIAGo++ and Stretch2, which are unable to move sideways. Corresponding templates sequence orientation, forward driving, and reorientation actions to replace Cartesian goals for driving.

The tuning effort required to achieve this transferability was minimal. MSCs themselves do not require any robot-specific tuning. The lMPC formulation depends on only two parameters, Δt and N (see Sec. 4). The former, Δt, is matched to the joint feedback rate and typically lies between 0.01s and 0.02s. Increasing N improves motion smoothness at the cost of increased computational effort. This parameter was selected as the minimum value required to accommodate robot-specific noise levels, typically between 15 and 30. For kinematic simulation, identical parameters were used across all setups, namely $\Delta t = 0.02$s and $N = 7$, in order to minimize computation time. The resulting level of transferability exceeds what can be achieved by single generative models without platform-specific fine-tuning.

5.3 Smooth Motion Task Transition

The dual UR10 setup was deployed in a sterility testing use case. One of the required motions was the insertion of a cylindrical object into a slot, corresponding to a classical peg-in-hole task. Using feature functions, an MSC template was defined in which the motion is decomposed into multiple phases, as illustrated in Fig. 8.

- **Always**: "Stay on Line", by which the distance between the cylinder bottom and the hole axis is minimized.
- **Approach**: "Reach Top", minimizing the distance between the cylinder bottom and a point above the hole, combined with "Slightly Tilted", enforcing that the angle between the cylinder and the horizontal world axis remains above a specified threshold.
- **Insert**: The tilt constraint is replaced by "Insert", formulated as a point-to-point goal.
- **Finish**: "Tilt Straight", aligning the cylinder axis with the hole axis.

The resulting execution demonstrates that task functions can be exchanged seamlessly during runtime. The measured trajectory shown in Fig. 7 exhibits smooth velocity profiles despite frequent task switches and the presence of sensor noise. This behavior empirically validates the proposed lMPC-based formulation for smooth motion task transitions.

6 Conclusion

This paper closes the motion execution gap by introducing MSC for symbolic motion composition, differentiable kinematic world models as semantic digital twins, and jerk-bounded lMPC for smooth task-function control. Validated on eight diverse robots like PR2 or Stretch2, it enables world-centric specifications, online monitoring, and robust transferability with guarantees unattainable by generative AI, which fails to enforce constraints or long-horizon consistency.

7 Limitations & Future Work

The framework currently lacks dedicated force control support. Basic force-aware behaviors can nonetheless be realized via monitors that react to sensor feedback and trigger MSC transitions, as demonstrated in the cutting example. The reason is that the kinematic world model does not yet represent robot dynamics, though QP-based controllers that handle such effects are demonstrated in related work [Bolotnikova *et al.*, 2021].

Finally, the system focuses on motion execution rather than planning, leaving integration with sampling based planners or trajectory optimizers as future work.

Acknowledgments

This work was partially supported by: (1) the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 Project-ID 329551904 "EASE - Everyday Activity Science and Engineering", University of Bremen (http://www.ease-crc.org/); (2) the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG); (3) the 'FAME' project, supported by the European Research Council (ERC) via the HORIZON ERC Advanced Grant no. 101098006. While partly funded by the European Union, views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[Aertbeliën and De Schutter, 2014] Erwin Aertbeliën and Joris De Schutter. etasl/etc: A constraint-based task specification language and robot controller using expression graphs. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1540–1546. IEEE, 2014.

[Andersson *et al.*, 2019] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[Beetz *et al.*, 2010] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1012–1017. IEEE, 2010.

[Beetz *et al.*, 2023] Michael Beetz, Gayane Kazhoyan, and David Vernon. The cram cognitive architecture for robot manipulation in everyday activities. *arXiv preprint arXiv:2304.14119*, 2023.

[Bolotnikova *et al.*, 2021] Anastasia Bolotnikova, Pierre Gergondet, Arnaud Tanguy, Sébastien Courtois, and Abderrahmane Kheddar. Task-space control interface for softbank humanoid robots and its human-robot interaction applications. In *2021 IEEE/SICE International Symposium on System Integration (SII)*, pages 560–565. IEEE, 2021.

[Borghesan *et al.*, 2015] Gianni Borghesan, Enea Scioni, Abderrahmane Kheddar, and Herman Bruyninckx. Introducing geometric constraint expressions into robot constrained motion specification and control. *IEEE Robotics and Automation Letters*, 1(2):1140–1147, 2015.

[Bouyarmane *et al.*, 2018] Karim Bouyarmane, Kevin Chappellet, Joris Vaillant, and Abderrahmane Kheddar. Quadratic programming for multirobot and task-space force control. *IEEE Transactions on Robotics*, 35(1):64–77, 2018.

[Bozcuoğlu *et al.*, 2018] Asil Kaan Bozcuoğlu, Gayane Kazhoyan, Yuki Furuta, Simon Stelter, Michael Beetz, Kei Okada, and Masayuki Inaba. The exchange of knowledge using cloud robotics. *IEEE Robotics and Automation Letters*, 3(2):1072–1079, 2018.

[Domínguez *et al.*, 2022] David Cáceres Domínguez, Marco Iannotta, Johannes A Stork, Erik Schaffernicht, and Todor Stoyanov. A stack-of-tasks approach combined with behavior trees: A new framework for robot control. *IEEE Robotics and Automation Letters*, 7(4):12110–12117, 2022.

[Escande *et al.*, 2014] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014.

[Fox and Long, 2003] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[Gupta *et al.*, 2024] Sthithpragya Gupta, Kunpeng Yao, Loic Niederhauser, and Aude Billard. Action contextualization: Adaptive task planning and action tuning using large language models. *IEEE Robotics and Automation Letters*, 2024.

[Harel, 1987] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[Haviland *et al.*, 2022] Jesse Haviland, Niko Sunderhauf, and Peter Corke. A holistic approach to reactive mobile manipulation. *IEEE Robotics and Automation Letters*, 2022.

[Huang *et al.*, 2025] Tzu-Yuan Huang, Armin Lederer, Dai-Jie Wu, Xiaobing Dai, Sihua Zhang, Stefan Sosnowski, Shao-Hua Sun, and Sandra Hirche. Sad-flower: Flow matching for safe, admissible, and dynamically consistent planning. *arXiv preprint arXiv:2511.05355*, 2025.

[Kim *et al.*, 2025] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[Kümpel *et al.*, 2025] Michaela Kümpel, Manuel Scheibl, Jan-Philipp Töberg, Vanessa Hassouna, Philipp Cimiano, Britta Wrede, and Michael Beetz. Everything robots need to know about cooking actions: Creating actionable knowledge graphs to support robotic meal preparation. *Frontiers in Robotics and AI*, Volume 12 - 2025, 2025.

[Mayr *et al.*, 2023] Matthias Mayr, Francesco Rovida, and Volker Krueger. Skiros2: A skill-based robot control platform for ros. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6273–6280. IEEE, 2023.

[Pan *et al.*, 2024] Tianyang Pan, Rahul Shome, and Lydia E Kavraki. Task and motion planning for execution in the real. *IEEE Transactions on Robotics*, 40:3356–3371, 2024.

[Pandala *et al.*, 2019] Abhishek Goud Pandala, Yanran Ding, and Hae-Won Park. qpswift: A real-time sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4):3355–3362, 2019.

[Pane *et al.*, 2021] Yudha Pane, Vahid Mokhtari, Erwin Aertbeliën, Joris De Schutter, and Wilm Decré. Autonomous runtime composition of sensor-based skills using concurrent task planning. *IEEE Robotics and Automation Letters*, 6(4):6481–6488, 2021.

[Röfer *et al.*, 2022] Adrian Röfer, Georg Bartels, Wolfram Burgard, Abhinav Valada, and Michael Beetz. Kineverse: A symbolic articulation model framework for model-agnostic mobile manipulation. *IEEE Robotics and Automation Letters*, 7(2):3372–3379, 2022.

[ROS.org, 2022] ROS.org. Urdf/xml. http://wiki.ros.org/urdf/XML, 2022. Accessed: 2023-05-07.

[Rovida *et al.*, 2018] Francesco Rovida, David Wuthier, Bjarne Grossmann, Matteo Fumagalli, and Volker Krüger. Motion generators combined with behavior trees: A novel approach to skill modelling. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5964–5971. IEEE, 2018.

[Sandakalum and Ang Jr, 2022] Thushara Sandakalum and Marcelo H Ang Jr. Motion planning for mobile manipulators—a systematic review. *Machines*, 10(2):97, 2022.

[Stelter *et al.*, 2022] Simon Stelter, Georg Bartels, and Michael Beetz. An open-source motion planning framework for mobile manipulators using constraint-based task space control with linear mpc. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1671–1678. IEEE, 2022.

[Stelter, 2025] Simon Stelter. *A Robot-Agnostic Kinematic Control Framework: Task Composition via Motion Statecharts and Linear Model Predictive Control*. PhD thesis, University of Bremen, 2025.

[Wächter *et al.*, 2016] Mirko Wächter, Simon Ottenhaus, Manfred Kröhnert, Nikolaus Vahrenkamp, and Tamim Asfour. The armarx statechart concept: Graphical programing of robot behavior. *Frontiers in Robotics and AI*, 3:33, 2016.