Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling

Conference Paper · October 2018

DOI: 10.1109/IROS.2018.8594319

5 authors, including:

Francesco Rovida
Aalborg University
11 PUBLICATIONS **90** CITATIONS
SEE PROFILE

Wuthier David
Aalborg University
6 PUBLICATIONS **20** CITATIONS
SEE PROFILE

Matteo Fumagalli
Technical University of Denmark
48 PUBLICATIONS **728** CITATIONS
SEE PROFILE

Motion Generators Combined with Behavior Trees: a Novel Approach to Skill Modelling

Francesco Rovida, David Wuthier, Bjarne Grossmann, Matteo Fumagalli and Volker Krüger

***Abstract*— Task level programming based on skills has often been proposed as a mean to decrease programming complexity of industrial robots. Several models are based on encapsulating complex motions into self-contained primitive blocks. A semantic skill is then defined as a deterministic sequence of these primitives. A major limitation is that existing frameworks do not support the coordination of concurrent motion primitives with possible interference. This decreases their reusability and scalability in unstructured environments where a dynamic and reactive adaptation of motions is often required. This paper presents a novel framework that generates adaptive behaviors by modeling skills as concurrent motion primitives activated dynamically when conditions trigger. The approach exploits the additive property of motion generators to superpose multiple contributions. We demonstrate the applicability on a real assembly use-case and discuss the gained benefits.**

Keywords: industrial robots, skills, reactive system, behavior trees, motion generators, assembly

I. Introduction

Despite recent efforts in automating a wide variety of industrial tasks, typical manufacturing scenarios still require manual work for highly repetitive operations. The problem arises with the stronger need for customization leading to smaller batch sizes: Common automated solutions are not profitable anymore due to their relatively long setup time with regard to their expected up-times. One such example is the engine assembly that is still today mostly performed manually. The research community is thus aiming at establishing robot programming methods for flexible manufacturing with some notable strengths being: 1) maximizing code reuse, 2) dealing with unstructured environments, 3) robustness to the occurrences of non-deterministic events and 4) intuitive languages, accessible to non-expert users.

A possible partial solution has been identified in encapsulating robot programs into parameterized robotic *skills* [1], realizing a semantically defined operation in the planning/user domain such as picking or placing an object, hence enabling skill-based software architectures to reconfigure production lines [2]. So far, the challenge of designing these skills has been tackled mainly from two fronts that are following antagonistic approaches: Bottom-up, with regard to control, which aims at capturing the continuous aspects of a task [3], [4], and top-down, with regard to coordination, which focuses more on the semantic aspects while often abstracting the control considerations [5], [6], [7], [8].

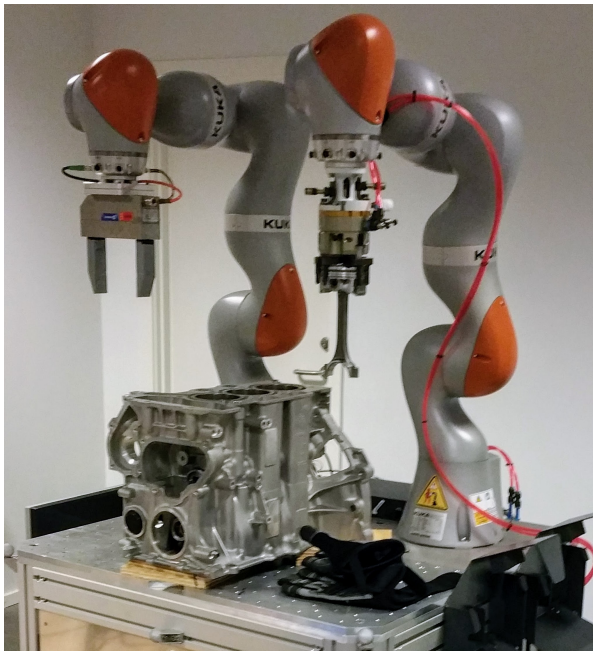


Fig. 1: Dual arm setup for the piston assembly use-case.

Current solutions that integrate control and coordination usually rely on *primitives* linked to static scripts or functions, such as iTaSC commands [3], encoding a continuous motion. The assumption is that primitives are going to be executed sequentially in order to not interfere with each other. However, we argue that a solution modeling combinable motion primitives can achieve higher modularity, better adaptation to unexpected situations and smoother transition from discrete action space to continuous motion. The solution requires (i) a coordination mechanism that can support *concurrent* primitives activation and (ii) a continuous motion description that can be divided into components and modified dynamically.

Among state-of-the-art solutions, one of the most popular coordination mechanism is the *Finite State Machines (FSMs)*: in [9], e.g., skills are modeled with UML/P statecharts and the *Task Frame Formalism* (TFF) [10] is used for motion description. In [11], and similarly in [12], each FSMs state contains *iTaSC*-based motion commands, extending the TFF with a more general way of defining constraints. In [13], skills are represented as a combination of a trajectory in pose-wrench space and a FSM describing the passage from a motion to another. In [7], skills are modeled with *Behavior Trees* [14], [15], that can be described as an evolution of

The research leading to these results has received funding from the European Commission's Horizon 2020 Programme under grant agreement no. 723658 (SCALABLE).

The authors are with the Robotics, Vision, and Machine Intelligence (RVMI) Lab at Aalborg University Copenhagen, 2450 Copenhagen, Denmark e-mail: francesco|daw|bjarne|m_fumagalli|vok@mp.aau.dk

*Hierarchical FSM*s, while each primitive defines a Cartesian motion command for the robot controller. In all the presented approaches, coordination of parallel motion primitives on the same resource is not considered.

In [16], we proposed a framework based on *extended Behavior Tree* (eBT). While being focused on automated planning, the method explored parallel execution of primitives, however, with the limitation of only coordinating independent primitives associated to different resources (e.g. opening the gripper while moving the arm, or planning next motion while executing another one). Inspired by Arkin et Al. [17] as an approach for cooperative coordination in behavior-based systems [18], this paper extends our eBT (and BTs in general) by proposing a method for executing motions in parallel, even when associated with the same resource (e.g. the same robot arm). We combine these motions by linking the eBT logic architecture with a configurable *Motion Generator* (MG). The eBT provides the coordination mechanism by activating or deactivating motion primitives based on their conditions, and collecting the contributions with a blackboard system. The MG is used to superpose individual, independent primitives to generate an hybrid, continuous motion.

The proposed approach has the advantage of modeling *modular* motion primitives that can be combined and adapted while executing a task in dynamic environments. Therefore, rather than providing strict rules for specific situations, it provides general *strategies* leaving freedom to the coordination system to select the best combination for the situation at hand. Furthermore, we claim the introduction of eBTs and MGs combinations as counterparts of *Dynamic Movement Primitives* (DMPs) [4]: DMPs and MGs both consists of attractor landscapes, where virtual springs attract the end effector toward a target position, virtual dampers slow down the movement and forcing terms generate deviations to the goal. However, DMPs' parameters are implicit, learnt from a set of demonstrations, whereas MGs' parameters are externally modulated by eBTs.

We demonstrate our approach by applying it on a challenging assembly task in which we progressively increase the number of unexpected cases the robot is able to deal with. The paper is structured as follows: section 2 introduces motion generators, section 3 presents the skill model and the coordination mechanisms, section 4 validates our approach by means of an experimental evaluation and finally section 5 draws our conclusions.

II. Motion generators

Our approach of generating arm motions in this work consists of controlling the end effector (EE) in Cartesian space. In particular, we require an *impedance controller* that fulfills the following criteria: 1) it should allow a generic varying Cartesian wrench to be superimposed over the motion and 2) to constrain velocities, accelerations and torques in order to respect common safety requirements. Any instance of impedance controller that provides these capabilities will, henceforth, be called *Motion Generator* (MG).

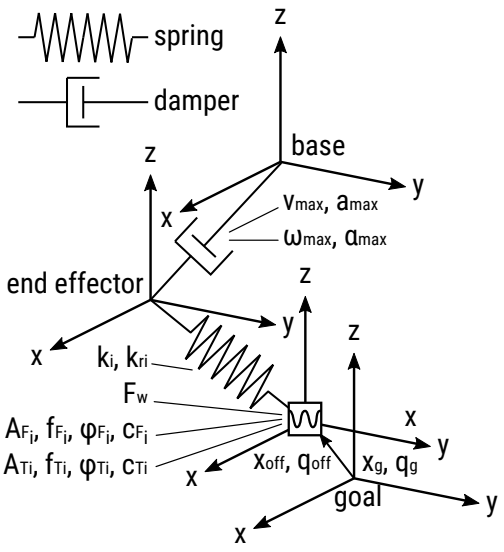


Fig. 2: Representation of the parameters of our motion generator as described in Tbl. I.

We assume MGs to target the following second-order dynamics for the EE:

$$m\dot{v} + bv = \bar{F_c} + F_d + F_w \quad (1)$$

$$\dot{x} = v, \quad (2)$$

where x is the position, v the velocity, m the apparent mass of the EE, b the damping, F_d the disturbances resulting from the interactions with the environment, F_w the feed-forward compensation for the workpiece's weight and $\bar{F_c}$ the saturated force such that

$$\bar{F_c} = \begin{cases} F_c, & \text{if } \|F_c\| \leq F_{\max} \\ F_{\max}\dfrac{F_c}{\|F_c\|}, & \text{otherwise} \end{cases} \quad (3)$$

$$\text{with} \quad F_c = -K(x - (x_g + x_{\text{off}})) + F_e, \quad (4)$$

with the maximum allowed force $F_{\max}$, the stiffness matrix $K = \text{diag}(\begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix})$, the goal position x_g, the offset over this goal position x_{off} and the excitation force F_e. In the present work, we chose sinusoidal signals with constant components as excitation forces:

$$F_e = \sum_{i=1}^{3} \big(A_i \sin(2\pi f_i t + \phi_i) + c_i\big) u_i, \quad (5)$$

with the amplitudes A_i, the frequencies f_i, the phase shifts ϕ_i, constant components c_i and the elementary vectors along axis i, e_i, $i = 1, 2, 3$.

m is assumed to be fixed and intrinsic to the robotic platform, whereas k_i, the maximum velocity $v_{\max}$ and the maximum acceleration $a_{\max}$ are specified as parameters. $F_{\max}$ and b can consequently be derived from a simple physical reasoning. From Eq. 1, assuming $F_d = F_w = 0$ N, b implies $v_{\max}$ as a limit velocity: $bv_{\max} = F_{\max}$. Also, the worse case scenario for the acceleration is $\|F_c\| = F_{\max}$, $\|v\| = v_{\max}$ and $bv = -\bar{F_c}$, yielding $m\dot{v} = 2F_c$. Thus, by

TABLE I: Summary of the parameters for the considered motion generator.

Parameter	Description
x_g, q_g	Position and attitude of the goal
x_{off}, q_{off}	Offset over the goal
k_i, κ_i	Linear and angular stiffnesses on axis i
F_w	Feedforward compensation of the workpiece's weight
A_{Fi}, A_{Ti}	Amplitude of oscilation on axis i (force and torque)
f_{Fi}, f_{Ti}	Frequency of oscillation on axis i (force and torque)
ϕ_{Fi}, ϕ_{Ti}	Phase shift of oscillation on axis i (force and torque)
c_{Fi}, c_{Ti}	Constant component on axis i (force and torque)
$v_{\max}$, $\omega_{\max}$	Maximum allowed linear and angular velocities
$a_{\max}$, $\alpha_{\max}$	Maximum allowed linear and angular accelerations

equalizing the norms, we get $F_{\max} = \frac{ma_{\max}}{2}$ and finally, by substitution, $b = \frac{ma_{\max}}{2v_{\max}}$. As an oscillatory behavior due to an under-damped system could be highly undesired, k_i should be chosen so that $k_i < \frac{b^2}{4m}$.

Note that the assumed rotational dynamics follows equivalent equations, which are not presented here for the sake of brevity. However, Fig. 2 and Tbl. I represent and summarize these parameters for both the translational and rotational dynamics.

By assuming default values for the parameters listed in Tbl. I so that $k_i > 0$ N/m, $k_{ri} > 0$ Nm/m, $v_{\max} > 0$ m/s, $\omega_{\max} > 0$ rad/s, $a_{\max} > 0$ m/s^2, $\alpha_{\max} > 0$ rad/s^2, $\|F_e\| = 0$ N and $\|x_{\text{off}}\| = 0$ m, MGs allow to express some motions in the TFF fashion [10]:

- A *guarded motion*, or compliant motion toward a surface until the detection of a contact through force feedback, can be expressed with $k_3 \leftarrow 0$ N/m and $c_{F3} \leftarrow F^*$, $\|F^*\| > 0$ N
- An *aligning motion*, with only $k_3 \leftarrow 0$ N/m
- A *screwing motion*, with $k_3 \leftarrow 0$ N, $k_{r3} \leftarrow 0$ Nm/rad, $c_{F3} \leftarrow c_F^* > 0$ N and $c_{T3} \leftarrow c_T^* > 0$ Nm

Our current representation of MGs therefore suffers from the same limitations as the TFF. However, an extension to ITaSC [3] support could overcome these limitations, but it is left for future works as this paper focuses on a proof-of-concept.

Other types of motions can also be expressed:

- A *withdrawal*, with $x_{\text{off}} \leftarrow x_{\text{off}}^*$, $\|x_{\text{off}}\| > 0$ m
- A *circular search strategy*, with $A_{F1}, A_{F2} \leftarrow A_F^* > 0$ N, $f_{F1}, f_{F2} \leftarrow f^* > 0$ s^{-1} and $\phi_{F2} \leftarrow \pi/2$ rad

The principle behind the superimposition of *arm motion* primitives is *orthogonality*: as we represent these primitives with subsets of MG parameters, two primitives with two non-overlapping subsets are *additive*. A simple example is the guarded motion superimposed with the circular search strategy.

However, since excitation signals belong to the wrench space, these could be directly summed if necessary.

III. Extended Behavior Tree

The *extended Behavior Tree* (eBT) model is our proposed approach to model a robotic skill that has been introduced in [16] in relation to task planning and execution optimization. eBT is an extension of the classical *Behavior Tree* (BT) model with concepts borrowed from *Hierarchical Task Networks* (HTN) [19] to describe, at the same time, how to execute a *task* and its effects on the world state. In this paper, we introduce briefly the model, while we give a detailed introduction to its execution aspects. For more details regarding the task planning process, we refer the interested reader to [16].

A. Overview

We define an eBT as a directed acyclic graph $G(V, E)$. In eBT each node can be either an abstract *procedure (P)*, a compound *skill (S)* or a skill *primitive (T)*. We refer to S and T also as implementations. In this paper, we will consider eBTs that have already been *expanded*, i.e. they do not contain any abstract procedure.

1) Skills: The skills' syntax is presented in Fig. 3. It contains the following terms:

- **type** - a unique symbol identifying the abstract procedure implemented by the skill.
- **name** - a unique symbol identifying the specific implementation.
- **params** - a set of parameters $x_1, \ldots, x_n$ required for planning and execution. Parameters can ground in world model's objects or in operation-space variables, but conditions can be applied only over objects. (see Sec. III-A.2)
- **pre-/post- conditions** - conditions that must be valid, respectively, just before and just after the execution. These are also used for the planning process. The possible conditions are described in Sec. III-A.3.
- **processor** - define the algorithm executed when the node is ticked. In addition to algorithms defined in [16], we added here the *Selector* [14].

Compound skills are inner nodes in the eBT, therefore they are also characterized by a list of children skills. Compound skills can boil down to nodes containing only a processor if they not define any parameters or conditions. Skill primitives are the leafs of the tree linked to specialized execution-functions. For brevity, primitives omit the processor in the notation and list their effects as post-conditions.

2) World model: The world model collects the general knowledge about the environment organized in a scene graph: a tree structure where nodes are objects and edges are relations of containment (`contain`) or membership (`hasA`) [20]. An *object* is defined as a tuple with: *id*, *type*, *label*

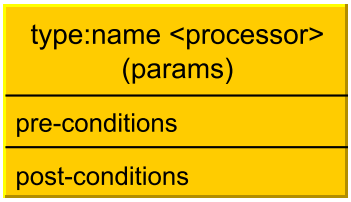


Fig. 3: Syntax of a skill.

and some *properties*. *Type* and *label* map the object to an individual in the ontology, *id* is a unique integer identifier of the object in the scene (abstract objects have id = -1, instantiated objects have id≥0) and *properties* is a list of tuples $(key, value)$, where key is a string, and $value$ is some data associated to the object, e.g. position, orientation or size. A relevant property of the scene graph necessary to understand the example given in the paper is that the pose of an object is always relative to the frame of the parent in the graph. For example, if a gripper is defined as a part of a robot object `(robot, hasA, gripper)`, the gripper pose will be considered relative to the robot frame.

3) Conditions: Conditions can be applied over and between world objects, to define a desired state to be valid before or after a skill's execution. We assume 5 possible conditions:

- isType(x_k, t): type(x_k) = t
- isGrounded(x_k): id(x_k) $\geq$ 0
- hasProperty(x_k, p): $\exists$ property(x_k)[p]
- property($\doteq$ x_k[p] value): x_k[p] $\doteq$ value, where $\doteq$ can be any comparison operator between $\{=, >, <, \leq, \geq\}$
- relation(x_k, predicate, x_j): $\exists$ [x_k, predicate, x_j] = True

For example, the condition `property(> src[WrenchNorm] 10)` requires the `WrenchNorm` property of the `src` object to be over 10 N. In some cases, to decrease the complexity to define conditions, can be convenient to use constraints in terms of qualitative relations, rather than quantitative constraints over properties. In our experiments, to define constraints on spatial relations between objects we use Allen intervals algebra. Allen intervals was born as an algebra for temporal intervals, but its application for spatial intervals was studied by [21]. Allen intervals define 13 possible relations between intervals: before (p), meets (m), equals (=), overlaps (o), during (d), starts (s), finishes (f) and their inverses. If the shape of objects is simplified into non-rotational cuboid, it is possible to relate objects spatially on each axis in terms of Allen intervals. Therefore we obtain 39 relations (13 Allen intervals times 3 axes). For example, the condition `relation(dst, dy, src)` requires the `src` object to be "during" (completely covered) by the `dst` object on the y axis.

B. Execution

The execution of an eBT is, analogously to BTs, done by propagating a *tick* signal from the root node down to the leaves. The visiting policy is defined by the skills' processor, but the order in our examples is always from left to right. When the tick signal arrives at a leaf (a primitive) of the tree, a function is called that performs a specialized operation and modifies the parameters' values. More than one primitive could be ticked at each iteration. When a primitive is ticked, it can return one out of three states: *success*, *failure* or *running*. The parent's processor decides how to handle the state, e.g. a *Selector processor* ticks the children in order and returns *success/running* as soon as a child returns *success/running*. In our notation, the condition

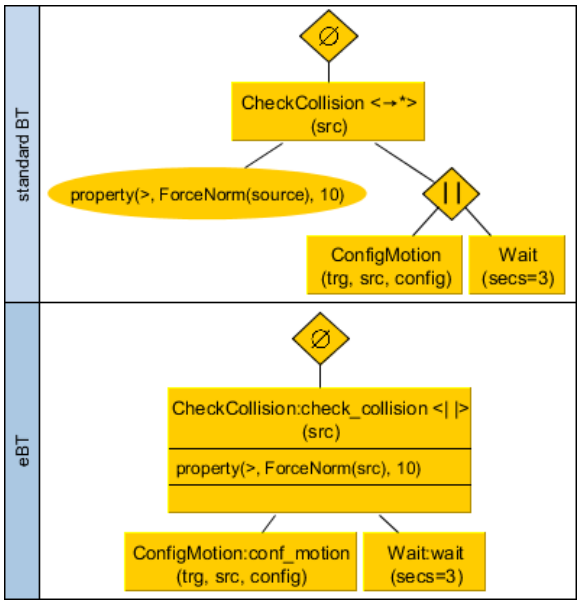


Fig. 4: A standard BT and an equivalent eBT representation.

nodes of standard BT are replaced by the skill's internal pre- and post-conditions.

If a skill is visited for the first time, the pre-conditions are checked. If pre-condition check fails, the skill is considered as it returned a failure. Pre-conditions replace the BT's condition nodes to achieve a more coherent integration with the planning domain. A nice side effect is that we obtain a more compact representation. In Fig. 4, we present a standard BT compared to an equivalent eBT. Since pre-conditions are checked only the first time a skill is activated, eBT results in an equivalent BT with one condition on the left, the rest of tree on the right, connected by a Serial node with memory (described in [14]). The memory effect is represented with the star (*) notation and means that the processor will avoid re-executing branches that already succeeded.

In the example, the `CheckCollision` skill is activated when the `src`'s `ForceNorm` property goes above a certain threshold. When active, the skill will continue to tick `ConfigMotion` and `Wait` until they both return success. If meanwhile the `ForceNorm` value goes below the threshold, it will be ignored. See Sec. IV for a more detailed description of `Wait` and `ConfigMotion`.

C. Blackboard

Sharing data without conflicts is one of the fundamental problems with concurrent programs. However, the synchronous nature of the tick signal greatly simplifies the problem with eBT. To collect and transport local information from skill to skill, we employ a blackboard system. The blackboard (BB) is a list of tuples $(key, value)$, where key is a string, and $value$ is some associated data. A single BB instance is associated to an eBT execution. Each time a skill receives the tick signal from the eBT, its parameters are updated with the values contained in the BB, and immediately after the ticking, the BB is updated with the

skill's parameters. The values in the BB that are linked to world objects are continuously synchronized with the world model in the background. There are some important points to remark. First, we do not distinguish between input and output parameters: all skill's parameters are pushed/pulled in the BB. Since the tick signal is synchronous we don't have any problem of concurrency when multiple skills are running: each skill will pull and push the values from and to the BB one at a time. As second consequence, the rightmost skill in the eBT will be the last to pull/push the values, becoming in fact the skill with highest "relevance" over the output of the eBT iteration, as it can override the output of previous skills.

D. Skill's coordination

One of the paper's contribution is a novel way to use the BT model to coordinate *motion skill primitives* - primitives imposing a motion on the robot. The usual way BT have been used in literature is to make a *competitive* coordination, that is when the activation of one motion excludes the activation of the others. In this paper, we propose to use BT also for a *cooperative* coordination, where the contribution of a motion primitive is blend together with other active ones to build up new hybrid behaviors. The key technical elements necessary to reach this coordination are:

- the parallel processor, to activate multiple primitives at the same time
- the blackboard system, to collect the contributions
- a MG as described in Sec. II, that can be reconfigured on-the-fly to generate hybrid motions

By modelling a task with the proposed approach, we observed several advantages:

- The competitive coordination as the fundamental idea of BTs allow us to cope with noise, inaccuracies or even sudden changes in the environment.
- The cooperative coordination allows us to easily define new complex motions such as shaking or screwing as superposition of simple ones.
- Again, exploiting cooperative coordination the integration of new behaviors such as constraints or error recovery can be easily accomplished without the need to completely rewire the tree in contrast to classical sequential methods such as FSMs. This indicates a good *scalability* of the approach.
- The use of the motion generator improves the modularity of the system and the reusability of skill primitives. In the example case, the entire behavior can be modelled with only 3 skill primitives using different parameterization.
- Being a finite-state model, it is possible to calculate every possible motion combination during the behavior specification, ensuring validity and safety.

IV. Application

The proposed framework has been applied to the challenging assembly use-case in which a piston has to be inserted into a cylinder of a motor block, as shown in Fig. 5. This is an instance of a typical peg-in-the-hole problem with the additional challenge of the freely swinging piston rod. In order to perform the task, the piston is placed in the robot's gripper beforehand and the motor block is placed on a table next to the arm with a known position of the target cylinder. The piston is then horizontally aligned with the cylinder and slowly moved downwards until the piston is inserted.

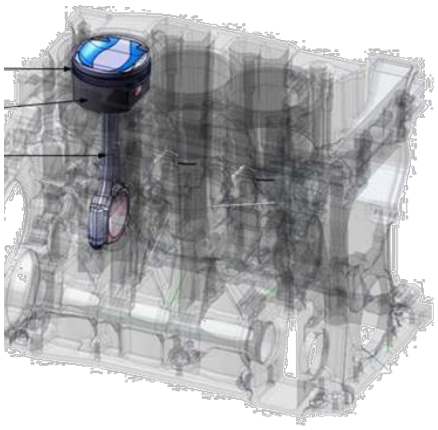


Fig. 5: Schematics of the piston inside the cylinder.

Due to the very tight fit of the piston and cylinder, this is not a trivial task. There are two common sources of errors (Fig. 7): First, the piston and the cylinder might be slightly misaligned resulting in a collision between the piston head and the cylinder. Second, the piston might have a larger misalignment or the piston rod is swinging due to the arm motion causing a collision between piston rod and cylinder/motor. In both cases, the piston cannot be correctly inserted without additional recovery behavior.

The task is performed by a *KUKA iiwa 7 R800* robot arm, and the software is deployed over a ROS [22] network based on a last generation desktop computer (i7 3.5 GhZ, 16 GB of RAM) and the robot's control cabinet. The communications are established through Gigabit Ethernet. The interface between ROS and the robot controller's native language is ensured by the iiwa stack from [23]. The MG consisted of the impedance controller at hand. Nevertheless, the factory commands for superimposing sinusoidal excitation signals were not used, due to their lack of generality. Instead, a module converting the desired wrench into position errors was implemented.

A. Motion configuration

In the core of our eBT, we require the definition of skill primitives that are related to particular arm motions. For the successful execution of the task, we identified four simple motions that can be easily expressed in terms of the same `ConfigMotion` skill primitive with different parameterization for all.

In the `Align` skill, the linear stiffness k_3 is zeroed. This removes the stiffness along the z-axis, i.e. the end effector is only attracted towards the target pose along the horizontal x-y-axes. The `Push` skill, in contrast, sets $c_{F3} = 80$ N,

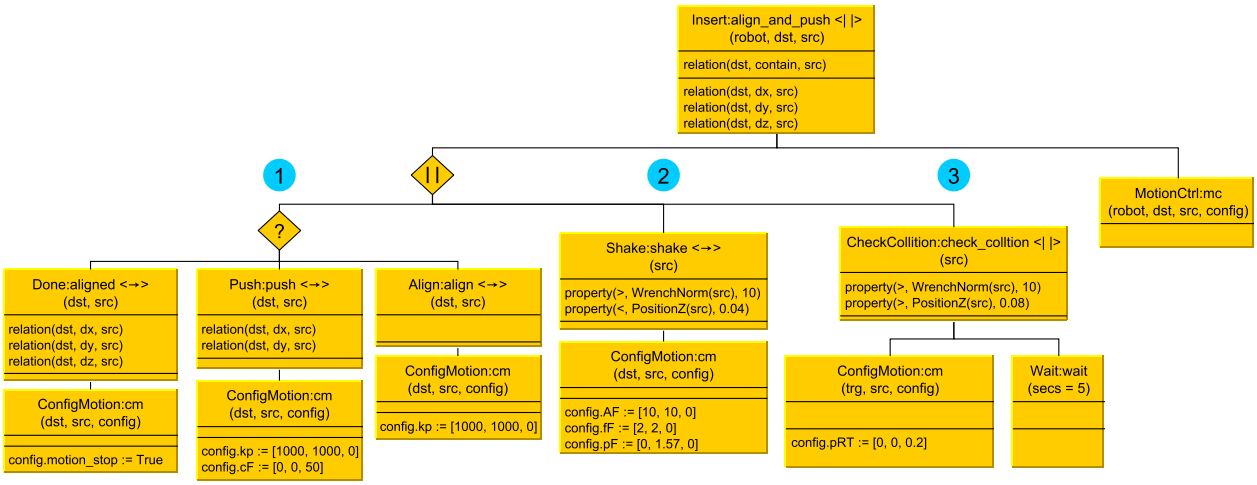
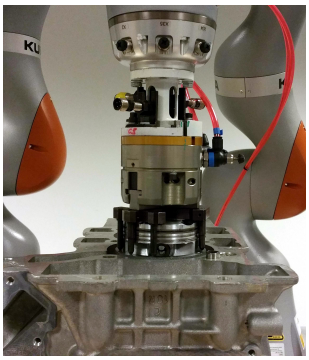
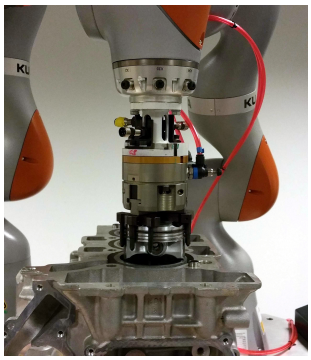


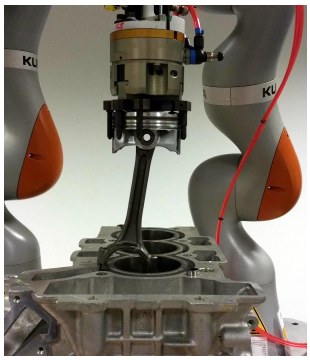
Fig. 6: The eBT used for the use-case. The three parallel branches add independently their contribution to the final motion configuration. Each branch implements a distinct behavior: Branch 1 defines the base behavior for the insertion by aligning and pushing. Branch 2 models a rotational motion, in cases where the piston head collides with the cylinder. Branch 3 represents an upwards movement with a short waiting time in cases where a collision between the piston rod and motor is detected.



(a) Case 1: No collision



(b) Case 2: Piston head collision



(c) Case 3: Piston rod collision

Fig. 7: The figures show three different cases considered in the piston insertion: (a) shows the case of a correctly inserted piston in the cylinder. In the second case (b), the piston is slightly misaligned causing the piston head to get stuck on the cylinder due to the tight fit. The last case (c) shows the collision between the piston rod and the motor block due to a larger misalignment or the swinging rod.

resulting in an constant progression along the vertical z-axis. The motion configuration of the `Shake` skill is slightly more complex: It sets $A_{F1} = A_{F2} = 10$ N, $f_{F1} = f_{F2} = 2$ Hz, $\phi_{F2} = 90°$. The three components result in a circular movement of the end effector (cf. II). The last configuration is set in the `CheckCollision` skill. Here, the attractor offset is set to $x_{\text{off}}^T = \begin{bmatrix} 0 & 0 & 0.2 \end{bmatrix}^T$ m, resulting in a new attractor position 20 cm above the task frame.

B. Behavior model

The whole assembly task is modeled within a single compound skill, namely the `Insert` skill, shown as the root node in Fig. 6. The `Insert` skill is divided into two parallel branches: The left branch configures the robot behavior using skills such as `Align` and `Push` by changing and accumulating motion parameters using the skill primitive `ConfigMotion`. The right branch consists of the single skill primitive `MotionCtrl` that is responsible for the generation and execution of the motion using the superimposed motion parameters. The eBT runs at a rate of 20Hz, meaning that the motion configuration is updated at the same rate (the low-level controller runs at a much higher frequency though).

To address potential errors during execution, we have specialized branches in the tree, that are *independent* from the base sequence of actions and add their contribution to the final robot behavior when specific preconditions are met. Here, we distinguish between three different cases (branches 1-3 in Fig. 6) according to the base sequence and the 2 most common errors:

Branch 1 defines the basic behavior to perform the insertion task with three skills: `Align`, `Push` and `Done`. The selector processor executes the first skill that meets

its respective pre-conditions: If the target position has been reached, the `Done` skill configures the motion generator to stop. Here, the alignment of gripper and target position is checked using the *Allen Intervals* algebra (cf. Sec. III-A.3). If the gripper is horizontally (x-y-axes) aligned with the target location, the `Push` skill configures the arm motion to vertically move towards the target location. In all other cases, the `Align` skill configures the arm motion to horizontally align with the target location.

Branch 2 considers the error handling for the collision of the piston head with the motor cylinder. Therefore, the `Shake` skill implements a search strategy for fitting the piston head into the cylinder by configuring the arm motion to make circular movements. It is activated when the sensed force norm is above a predefined threshold of 10 N and the vertical distance to the target position is less than 4 cm.

Branch 3 adds error handling for the collision of the piston rod (or other parts) with the motor cylinder. Therefore, the `CheckCollision` has been implemented that recovers the piston from a stuck position by simply inducing an upwards movement until the piston is 20 cm above the target position again At the same time, the skill runs a `Wait` skill primitive for 5s to keep the node active (cf. Sec. III-B) and to let the piston rod stabilize again before continuing with the base behavior. Similar to the branch 2, it is only activated if the sensed force is above 10 N while, in this case, the vertical distance to the target position is larger than 8 cm.

C. Evaluation

In order to evaluate the implemented behavior model, we conducted several experiments while incrementally adding the previously mentioned error cases. The experiments can be summarized with three distinct cases A, B and C which we describe and analyze in the following:

For **experiment A**, we solely use the first branch to insert the piston into the cylinder. The task can only be executed successfully if the target position and the piston are almost perfectly (up to a deviation of ~1 mm) aligned during insertion such that no collisions occur. Figure 8a shows the time series of the insertion when using an almost perfect target: The first stage (0-120 timesteps) reflects the horizontal alignment phase in which the error on the x-y-axes converges to 0 m. In the second stage (120-200 timesteps), the piston is vertically pushed towards the target position. We can see that due to the tight fit of the piston head and cylinder, the measured force rises up 20 N for a short moment, however, the piston immediately slips into the cylinder to its final destination.

For **experiment B**, we added the second branch (combined branch 1 and 2). The error recovery behavior for collision between piston head and cylinder allow us to deal with slightly larger alignment errors up to ~2 mm. The collisions are detected correctly and the superimposed `Shake` skill leads to a successful insertion of the piston. This behavior can cope with alignment errors up to 2 mm, since larger deviations lead to an earlier collision between the piston rod and the motor block. Figure 8b shows the time series of the insertion when the piston head and the cylinder are colliding: The horizontal alignment and vertical alignment stages (0-200 timesteps) are similar to case 1, however, in contrast to the previous case, the piston head does not slip into the cylinder. We can see that the z-position stabilizes at around 2 mm while the gripper experiences a constant high force, indicating that the piston head is stuck at this location. The high force then enables the `Shake` skill (220 timesteps onwards) that superimposes a circular movement on the `Push` skill causing a sinusoidal movement of the x- and y-coordinates while still pushing downwards. Finally after 260 timesteps, the piston head reaches the correct position and slips into the cylinder.

For **experiment C**, we use all three branches (combined branch 1, 2 and 3). In addition to the error handling of branch 2, we introduce the collision check for the piston rod. The superimposed `CheckCollision` skill is able to compensate for collisions due to a swinging rod or a misalignment of up to 5 mm. Figure 8c shows the time series of the insertion when the piston rod and the cylinder are colliding: After the horizontal alignment (0-100 timesteps), the `Push` skill tries to insert the piston into the cylinder, however, due to the swinging rod or a large misalignment, the rod gets stuck on the side of the cylinder. This causes an increased force on the gripper (150-170 timesteps) leading to a slight misalignment on the y-axis. When the measured force exceeds 10 N, the `CheckCollision` skill is triggered: the gripper is moved to a predefined height offset (20 cm) from the target position and then remains in that position till the 5 s waiting time (`Wait` skill primitive) is over. Afterwards, the behavior switches back to its normal behavior: The piston is pushed downwards (250-350 timesteps) and the `Shake` skill is enabled (370 timesteps) to correct for the small misalignment. After 450 timesteps, the piston is correctly inserted in the cylinder.

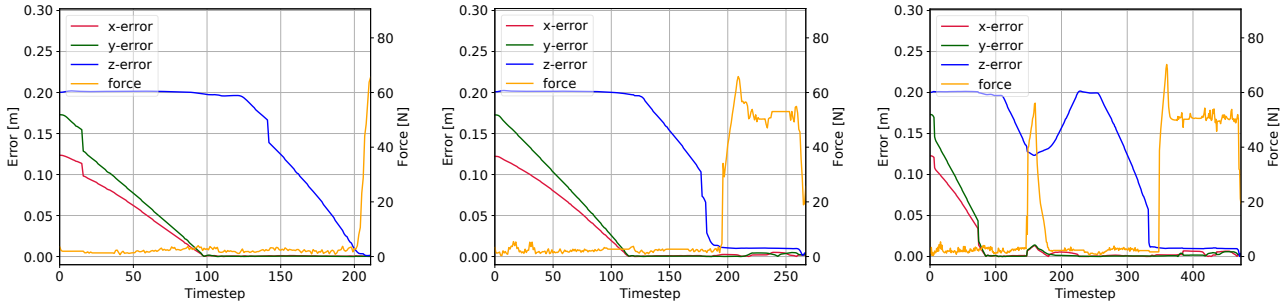
The experiments show that the proposed behavior model using the eBT implementation is in fact able to successfully perform the assembly use-case, even with the introduction of misalignment errors in the system. The results are summarized in Table II.

TABLE II: Increase of robustness over cylinder misalignment when executing only branch 1, branches 1 and 2 together and all branches.

Experiment	0-1 mm	1-2 mm	2-5 mm
A (Branch 1)	✓	✗	✗
B (Branch 1, 2)	✓	✓	✗
C (Branch 1, 2, 3)	✓	✓	✓

V. Conclusions and Future Work

We presented a novel skill-based framework for robot programming based on motion generators that superpose the contribution of parallel motion primitives to generate a robot behavior adaptable to dynamic environments. Within this framework, the skill developer can program a task



(a) Experiment **A**: No collision (b) Experiment **B**: Piston head collision (c) Experiment **C**: Piston rod collision

Fig. 8: Time series of the three different experiments. 1 second is approximately 40 timesteps which we prefer to use due to a fluctuating message frequency in the recordings. (a) shows the time line of the insertion without major collisions. (b) shows the case in which the piston head is colliding with the cylinder. The `Shake` skill is activated shortly after (~200 timesteps) and resolves the error situation. (c) additionally shows the case in which the piston rod is colliding with the cylinder and is getting stuck. The `CollisionCheck` skill is activated to recover from the error (~160 timesteps). Additional collision between the piston head and cylinder is then handled by the `Shake` skill again (~350 timesteps).

together with non-choreographed recovery strategies. Gained benefits are the skills' modularity and scalability to complex environments by decoupling recovery strategies from the main task. Preliminary tests conducted on a motor assembly showed that the approach has potential to solve real industrial use-cases. However, in order to create adaptable behavior for general tasks, future work should analyze the transformation from specification to resulting motions more in-depth. Furthermore, it would be of high interest to explore the integration of additional temporal and spatial constraints within the system, besides the ones imposed on the target pose.

REFERENCES

[1] S. Bøgh, O. S. Nielsen, M. R. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?" in *The 43rd Intl. Symposium of Robotics (ISR)*, 2012.

[2] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. Petrick, and V. Krüger, "SkiROS-A skill-based robot control platform on top of ROS," in *Studies in Computational Intelligence*, 2017, vol. 707, pp. 121–160.

[3] J. D. Schutter and T. D. Laet, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Journal of Robotics*, pp. 1–33, 2007.

[4] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in neural information processing systems*, 2003, pp. 1547–1554.

[5] J. Huckaby and H. Christensen, "A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics," *Proc. 26th AAAI Cognitive Robotics Workshop*, pp. 94–101, 2012.

[6] S. Balakirsky, Z. Kootbally, T. Kramer, A. Pietromartire, C. Schlenoff, and S. Gupta, "Knowledge driven robotics for kitting applications," in *Robotics and Autonomous Systems*, vol. 61, no. 11. Elsevier B.V., 2013, pp. 1205–1214.

[7] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6167–6174, 2015.

[8] M. Crosby, F. Rovida, V. Krüger, and R. Petrick, "Integrating Mission and Task Planning in an Industrial Robotics Framework," *International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

[9] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A New Skill Based Robot Programming Language Using UML / P Statecharts," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 461–466, 2013.

[10] H. Bruyninckx and J. D. Schutter, "Specification of force-controlled actions in the task frame formalism-a synthesis," *Robotics and Automation, IEEE*, 1996.

[11] R. Smits, "Skills: Design of a Constraint-Based Methodology and Software Support (Phd) (Robot vaardigheden: Ontwerp van een beperkingsgebaseerde methodologie en software)," Ph.D. dissertation, 2010.

[12] M. Linderoth, "On robotic work-space sensing and control," Ph.D. dissertation, Lund University, 2013.

[13] A. Wahrburg, S. Zeiss, B. Matthias, J. Peters, and H. Ding, "Combined pose-wrench and state machine representation for modeling Robotic Assembly Skills," *IEEE International Conference on Intelligent Robots and Systems*, vol. December, pp. 852–857, 2015.

[14] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ogren, "Towards a Unified Behavior Trees Framework for Robot Control," *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[15] M. Colledanchise and P. Ogren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2017.

[16] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6793–6800, 2017.

[17] R. C. Arkin, "Towards the Unification of Navigational Planning and Reactive Control," *Proc. Am. Assoc. Artif. Intell.*, vol. Spring Symp. Robotics Navig., pp. 1–5, 2001.

[18] ——, *Behavior-based Robotics*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[19] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*, ser. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, 2004.

[20] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, 2015.

[21] M. Mansouri and F. Pecora, "A Representation for Spatial Reasoning in Robotic Planning," *IROS 2013 Workshop: AI-based Robotics*, 2013.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[23] C. Hennersperger, B. Fuerst, S. Virga, O. Zettinig, B. Frisch, T. Neff, and N. Navab, "Towards mri-based autonomous robotic us acquisitions: A first feasibility study," *IEEE transactions on medical imaging*, 2016.