

Skill-based Programming Framework for Composable Reactive Robot Behaviors

Yudha Pane, Erwin Aertbeliën, Joris De Schutter and Wilm Decré

Abstract—This paper introduces a constraint-based skill framework for programming robot applications. Existing skill frameworks allow application developers to reuse skills and compose them sequentially or in parallel. However, they typically assume that the skills are running independently and in a nominal condition. This limitation hinders their applications for more involved and realistic scenarios e.g. when the skills need to run synchronously and in the presence of disturbances. This paper addresses this problem in two steps. First, we revisit how constraint-based skills are modeled. We classify different skill types based on how their progress can be evaluated over time. Our skill model separates the constraints that impose task-consistency and the constraints that make the skills progress i.e. reaching their end conditions. Second, this paper introduces composition patterns that couple skills in parallel such that they are executed in a synchronized manner and reactive to disturbances. The effectiveness of our framework is evaluated on a dual-arm robotics setup that performs an industrial assembly task in the presence of disturbance.

I. INTRODUCTION

The demand for mass customization has forced manufacturing processes to be more agile than ever before. As a consequence, robotic systems, as one of the key enabling technologies, need to be more configurable and adaptable to a wide range of situations. Furthermore, since many aspects from the environment cannot be modeled accurately, in many cases robots also need to be **reactive** with respect to unforeseen disturbances. That is, utilizing online information e.g. sensor measurements to adapt their trajectory in order not to violate constraints such as collision avoidance.

Programming a complex reactive behavior requires a different approach than the traditional methods of simply specifying point-to-point (PTP) or impedance motion. This is especially important because robot programming is often performed by *application developers* who typically have less expertise in robotics. They also need to develop applications in a shorter time, thus programming with simple motion primitives is too laborious and undesirable.

One of the solutions proposed in the literature is skill-based robot programming [1] [2] [3] [4]. Skills give robots the capability to perform a certain behavior e.g. trajectory following or force-based insertion. Skills are implemented by encapsulating the underlying motion controller and exposing a set of configurable parameters to the users. This way,

*This work was supported by Flanders Make SBO PROUD and MUL-TIROB

All authors are with the Dept. of Mechanical Engineering division RAM, KU Leuven, Celestijnenlaan 300, Heverlee 3001, Belgium, and members of Core Lab ROB, Flanders Make.

Corresponding author: Yudha Pane. e-mail: yudha.pane@kuleuven.be

application developers can easily reuse the skills without having to deal with writing the reactive behavior themselves.

In this paper's context, we define the term **composability** as the ability for application developers to compose existing skills in order to yield a predictable combined and desired behavior. One of the main limitation of the current skill-based programming frameworks is their limited composability. This in turn reduces the reusability of existing skills. This problem is partly due to the limitation of the underlying skill specification and the relatively rudimentary composition patterns i.e. the possible ways of composing skills. For instance, existing frameworks allow application developers to specify skills and compose them to be executed concurrently or sequentially. But they do not facilitate to specify more complex composition such as imposing synchronization constraints between concurrent skills or composing skills to handle non-nominal situations such as collision or human-induced disturbances.

In this paper, we present a new skill programming framework based on a **constraint-based approach**. Our framework differs from existing solutions because it not only allows the programmers to change the skill parameters, but also allows them to apply and specify composition constraints to create coupled behavior between the skills. These composition constraints are presented as patterns; they are semantically well-defined, thus can be understood and reused by application developers. The contributions of this paper are: *i)* a constraint-based skill model that separates the *task* and *progress* constraints, *ii)* reusable composition patterns for composing skill behaviors in non-trivial manners, *iii)* evaluation of the proposed framework in the context of dual arm robotic assembly task.

The remainder of the paper is organized as follows. In Section II, we present an overview of the previous work. Section III describes how we model constraint-based skills. The different composition patterns are presented in detail in Section IV. We evaluate the framework in an industrial assembly task and report the results in Section V. Section VI concludes the paper.

II. RELATED WORK AND MOTIVATION

Methods to generate real-time, reactive robot motion while executing a nominal task constitute an extensive research area within the robotics community. A variety of techniques have been proposed to address different applications. In the context of collision avoidance behavior, approaches based on potential function are one of the most widely-used techniques [5]. Efforts to combine local and global planning result in a

mixed method, such as elastic strips [6]. In [6], it is possible to compose nominal behavior with non-nominal ones such as following path while avoiding obstacle and maintaining posture by assigning priority levels.

Meanwhile, in the context of reacting to a contact situation, constraint-based techniques such as the task frame formalism (TFF) are more often used [7]. This formalism allows us to specify force/torque-controlled tasks along and around independent task frame axes. More recently, there has been significant progress in constraint-based programming, resulting in, among others, the iTaSC [8] [9], eTaSL [10], and stack-of-tasks (SoT) frameworks [11].

The iTaSC framework can explicitly model geometric uncertainty to improve robustness of the task execution with respect to disturbances. It also proposes the use of feature coordinates to enable more general constraint expressions. However, with iTaSC, the programmer is forced to formulate a six DoF virtual kinematic chain (VKC). This restriction is not necessary in eTaSL, thus the singularity problem inherent with the VKC can be avoided. In eTaSL, the separation of concerns between the task specification and the solver is properly addressed, such that the same specification can be executed with different types of solver. The SoT is an alternative framework that differs from eTaSL and iTaSC in the way the optimization problem is solved. In particular, a null-space projection technique is used such that lower priority constraints will not affect the higher priority ones.

One main advantage of the constraint-based approach is the composability of constraints. Programmers can develop robot skills by composing constraints instead of composing trajectories. When the constraints are conflicting, different weights or priority levels can be assigned. Another advantage is that, in most cases, the skills can be executed in real-time, and therefore are suitable for industrial robot tasks.

Due to its versatility, the constraint-based approach is adopted as the underlying method in developing skill-based programming frameworks. For example, in [2], Thomas et al. propose LightRocks, a domain specific language (DSL) for programming sensor-based skills. The skill is specified based on an extended TFF. The programmers can compose skills using the UML/P state chart model.

Another example is the framework developed by Nägele et al. [4] [12] where the skill specification is based on iTaSC. It facilitates to instantiate new skills from already existing instances by following the prototype pattern. More complex skills can be composed hierarchically, both in sequential and parallel manner.

Other frameworks achieve flexibility using path planning or ad hoc motion generation instead of a constraint-based reactive approach. This approach can achieve a considerable amount of flexibility but with a slower reactivity. An additional drawback is a reduced composability between skills since they do not share a common underlying specification. One example is proposed by Pedersen et al. [1], who showed that skills can either be executed using a TFF-based controller or by an off-the-shelf motion planner [13]. The framework allows non-experts to compose skills sequentially,

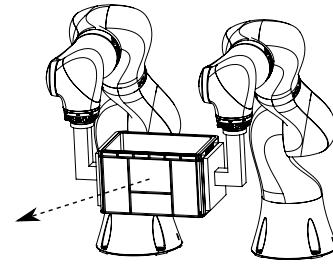


Fig. 1. Cooperative object transfer with dual arm setup

forming a task. A transition occurs when the post-conditions of the source skill and the pre-conditions of the target skill hold. Another example is from Stenmark et al. [3] [14]. Their work does not particularly focus on composability of skills, but rather on how to parameterize the skills in an intuitive way. They also evaluated their prototype tool for programming dual-arm tasks by non-robot experts [14].

The above mentioned frameworks reduce the effort in skill programming by abstracting away the low-level coding. However, the relatively simple composition rules limit the level of complexity that can be achieved for the robot behavior. In our approach, composing two skills does not simply mean merging their constraints, but also introducing additional constraints that describe the coupled behavior between the composed skills. This is particularly relevant when the execution of skills is subject to non-nominal conditions or disturbances. For example, consider the problem of transferring a box with two robots as illustrated in Figure 1. Each robot executes a sequence of skills to approach the box and bring it to a target pose. In the nominal situation, the two skills can be executed independently in parallel. However, in a non-nominal situation such as in the presence of human-induced disturbance, each robot should adapt its skill's progress to maintain a proper grip. Here, the simple composition strategy will fail. Alternatively, one may argue that such a problem can be solved by a single skill that controls both robots, forming a closed kinematic chain with the object. However, this implies the need to create a skill that is application-specific, hence not so reusable. The perspective of this research is to provide application developers with a set of generic skills, while facilitating the creation of more complex skills by means of compositions.

III. CONSTRAINT-BASED SKILL MODELLING

This section presents the underlying model that describes how a constraint-based skill is specified and structured. The model serves as a guideline for *skill developers* i.e. robotics experts who implements skills. The developed skills can then be configured and reused by the application developers who are the domain-experts. This way, the application developers can focus on defining the tasks at hand and selecting the suitable skills. In this work, we adopt eTaSL to specify the skill's constraints. This choice is due to its strong separation of task specification and execution and its relatively mature development. Furthermore, one of the advantages of eTaSL is

the support of feature variables. These are auxiliary variables that are used to specify degrees of freedom in the skill. Such variables are useful not only for specifying how a skill progresses, but also how it deviates from the nominal path. We explain this in more detail below.

A. Constraint and Monitor

A skill is mainly composed of a set of *constraints* and *monitors*. The constraints and monitors are acting upon *expressions*. The expressions can depend on *input* signals such as sensor values. These expressions can also be specified as *outputs*. The skill developers typically spend most of their effort in specifying and testing these two components. Ideally, the developed skill should represent a well-defined reactive robot behavior, specifying not only the nominal behavior but also how to deviate in case of disturbances. This way, it is easier for the application developers to predict the resulting motion and select suitable skills for their use cases.

In eTaSL, a constraint is imposed on a kinematic expression e :

$$e = f(q, \chi_f, t) \quad (1)$$

where $q \in \mathbb{R}^{n_r}$ denotes the n_r joint variables, $\chi_f \in \mathbb{R}^{n_f}$ denotes the n_f feature variables and $t \in \mathbb{R}$ is time. Skill developers can specify a constraint by imposing target values (equality) or bounds (inequality) on the expression either in position or velocity level.

In a more formal notation, a constraint can be represented as a tuple $\mathcal{C} := \langle e, e_d, C_r \rangle$ where e is the kinematic expression that we want to constrain, e_d is the desired value/bounds for the expression, and C_r is the assigned controller. The target e_d may consist of the feedforward and feedback terms $e_d = \{\dot{e}_d^{\text{ff}}, \dot{e}_d^{\text{fb}}\}$. In eTaSL, the controller steers e such that it converges with the following dynamics:

$$\dot{e} = \dot{e}_d^{\text{ff}} + K_c(\dot{e}_d^{\text{fb}} - e) \quad (2)$$

The control gain K_c can be set to obtain a desirable time constant.

Monitors continuously evaluate an eTaSL expression and trigger events once certain conditions are met, e.g. when the pose error between is lower than a given threshold. They are useful for end conditions of skills, such that transition to the subsequent skills can be performed.

From the point of view of the application developers, skills are simply configurable and composable building blocks. The detailed implementation of the skill-specific constraints and monitors are abstracted away from them. Instead, they are exposed to the constraints and monitors' parameters, which can be adapted to suit the target application.

B. Specifying Skill Behavior with Constraints

Constraints are the main component of a skill as they determine the robot's overall behavior. This work makes a distinction to skill's constraints into either *task* or *progress* constraint. Task constraints result in a skill behavior of staying in the task space. Meanwhile, the progress constraints result in a behavior of advancing skill, i.e. it drives the

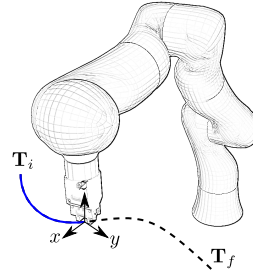


Fig. 2. A Cartesian motion skill that moves a robot end-effector to a target pose while following a given path (dashed line). The blue solid line signifies the path already travelled.

Skill Specification	
Task Constraints : \bar{w}_t	Progress Constraints : \bar{w}_p
$c_{t_1} : w_{t_1}, p_{t_1}$	$c_{p_1} : w_{p_1}, p_{p_1}$
$c_{t_2} : w_{t_2}, p_{t_2}$	$c_{p_2} : w_{p_2}, p_{p_2}$
\vdots	\vdots
$c_{t_n} : w_{t_n}, p_{t_n}$	$c_{p_m} : w_{p_m}, p_{p_m}$

Fig. 3. The structure of constraints in a skill specification

skill from its initial to goal condition. Together, these two constraint classes result in a robot that performs the skill-specific behavior while trying to get closer to its goal state. As an illustration, consider a path following skill as depicted in Figure 2. The skill moves the robot end-effector from initial pose T_i to a target pose T_f . Here, the task constraints will keep the robot end-effector on the path. At the same time, the progress constraint will enforce the end effector to move along the path by following a given progress profile. This separation of constraint types in a skill specification is in contrast to previous works that treat all task constraints uniformly. Such a distinction is beneficial since it introduces a degree-of-freedom within the task space. The skill can optimize this degree-of-freedom to resolve non-nominal situation such as the presence of disturbances or conflicting objectives. Furthermore, it also allows us to compose skills by synchronizing their progresses, as will be shown in the subsequent section.

The task and progress constraints can have different weights and priorities. If the task constraints' weight is larger than the progress constraints' weight, then the robot prefers to remain in the task space and sacrifices its progress whenever a disturbance occurs. Conversely, when the progress constraints have larger weight or higher priority, then the robot prefers to stay on track with its nominal progress while deviating from its task space. This way, different deviating behavior can be achieved by varying the two constraints weights or assigning different priority levels to them.

Figure 3 outlines the constraints structure in a skill specification. The skill may be composed of n task constraints and m progress constraints. Each task constraint $c_{t_i} \in \mathcal{C}$ has its own weight w_{t_i} and priority p_{t_i} . A constraint c_{t_i} will not violate c_{t_j} when $p_{t_i} > p_{t_j}$, regardless of their weights. The weights difference will only have an effect when the two constraints are in the same priority level. The relative

importance between the task and progress constraints can be adjusted by setting \bar{w}_t and \bar{w}_p . These weight factors will be multiplied with the weight of each task and progress constraint, respectively. This way, the relative importance within the same constraint group remains the same.

C. Modelling Skill Progress

A progress constraint is imposed on *progress variable*, $s = [0, 1]$, which is a normalized scalar expression that measures the skill's advancement in continuous time. The skill starts when $s = 0$, while $s = 1$ implies the skill has reached its end goal. This paper assumes that s is an increasing function in the nominal situation i.e. in the absence of disturbances. As an example, consider again the path-following skill of Figure 2. Assuming the robot tracks the path by following a target pose generated by a motion profile, the progress variable is simply $s = \frac{\chi_{f_{pv}}}{L}$ where $\chi_{f_{pv}}$ denotes the coordinate along the path and L denotes the path length. A progress constraint can be specified such that s increases with a constant speed. As the motion profile is a function of the path coordinate $\chi_{f_{pv}}$, the task constraints will enforce the robot end-effector to track the generated pose while satisfying the constant progress speed in a nominal situation. In general, the progress variable of a given skill can be mathematically expressed as a function of the joint and feature variables $s = f(q, \chi_f)$.

Although the separation of task and progress constraints is useful to achieve different behaviors, specifying a progress constraint is not applicable to all skills. This is because not every skill has an explicit measure of advancement. For example, skills that solely performs reactive behavior upon disturbance such as collision avoidance or admittance motion do not actually "make progress" as time elapses. For these types of skill, progress constraints do not exist.

D. Classifying a Skill Based on its Progress

As different skills produce different behaviors, how their progresses are defined may be different. Based on the property of their progress variables, we analyze and classify different types of skills. Such classification is needed in determining whether two skills can be composed together using the patterns proposed in Section IV.

The different skill types along with their examples are summarized in Table I. The left column (Type-I) describes the first category of skills whose progress can be expressed as a function of the optimization variables q and χ_f . We say that their s values are instantaneously *observable* and *controllable*. The second group (Type-II) are skills that, in nominal condition, have increasing progresses, but the values are neither observable nor controllable. These skills typically finish upon detecting certain events. An example is guarded motion, i.e. a skill that approaches an object with unknown location and stops as soon as contact is made. As the skill runs, it becomes closer to the object, thus the progress increases. But due to the unobservable end state, its value can not be quantified. Finally, the last group (Type-III) belongs to skills that do not have defined progresses, therefore lack the

progress constraints. These skills typically do not produce any motion in a nominal condition, but perform reactive motion in the presence of disturbance. An example is an admittance motion skill where the robot only moves when force is applied. Since these types of skills do not make progress by themselves, they are commonly composed in parallel with progressing skills.

IV. COMPOSING BEHAVIORS WITH COMPOSITION PATTERNS

More complex reactive behaviors can be programmed by means of composing skills. This paper focuses more on composability in continuous level while assuming that the discrete-level composition can be simply specified with well-known coordination models such as finite-state machine (FSM) or behavior trees.

We consider more involved parallel behaviors in which the composed skills are not just running independently, but they are governed by coordination schemes, namely *composition patterns*. Each pattern results in a different composition behavior with well-defined semantics such that application developers can reuse them for their target applications.

Formally, a composition pattern CP can be viewed as a function that maps two skills into a new composed skill:

$$CP : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S} \quad (3)$$

where \mathcal{S} denotes the skill set. Application developers can create increasingly more complex behaviors by composing multiple patterns. For example, given two different composition patterns CP_1 and CP_2 , more complex compositions can be created:

$$CP_1 \circ CP_2 : (\mathcal{S} \times \mathcal{S}) \times \mathcal{S} \rightarrow \mathcal{S} \quad (4)$$

A. Progress Synchronization Patterns

These patterns are motivated by tasks that require time-domain synchronization of robot behaviors. In such tasks, the active skills not only run simultaneously, but they have to reach their goals in a synchronized manner. In general, these patterns handle such problems by imposing constraints on the skills' progress variables. Here, we propose two synchronization patterns.

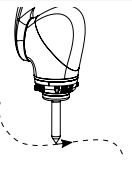
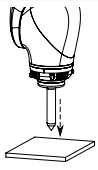
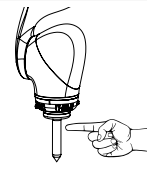
1) *Master-Slave Synchronization*: This mode is useful when a task requires one skill to follow another's progress. The skill that is assigned a "master" role may progress independently with respect to the "slave" skill. When the master skill is perturbed, the slave skill has to adjust its progress variable accordingly to maintain a synchronized motion. In eTaSL, the constraint that realizes this pattern is imposed on the slave's progress variable:

$$c = \langle e = s_{\text{slave}}, e_d = \{\dot{s}_{\text{master}}^{\text{ff}}, s_{\text{master}}\}, C_{\text{proportional}} \rangle \quad (5)$$

where $C_{\text{proportional}} \in \mathcal{C}_r$ is a controller that satisfies (2). This pattern can be viewed as an operation that is applied to two skills $S1, S2 \in \mathcal{S}$:

$$\text{ComposeMasterSlave} : S1 \times S2 \rightarrow S3 \quad (6)$$

TABLE I
DIFFERENT SKILL CLASSIFICATION BASED ON THEIR PROGRESS VARIABLE PROPERTIES

Type-I: Skills with controllable progress	Type-II: Skills with uncontrollable progress	Type-III: Non-progressing skills
 <ul style="list-style-type: none"> • The progress variable is instantaneously observable and controllable • Varying the progress variable moves the skill along its task space • Example: Cartesian motion with motion profile 	 <ul style="list-style-type: none"> • The progress variable can not be instantaneously determined and controlled • Reaching the goal condition is only known upon an event detection • Example: guarded motion 	 <ul style="list-style-type: none"> • The progress variable can not be defined • Complement the progressing skill to actually achieve a task • Examples: admittance, collision avoidance

where the master, S1, can be either Type-I or Type-II skill while the slave, S2, is a Type-I skill to ensure controllability. This operation results in a composite skill, S3, which inherits the master skill's type.

2) *Peers Synchronization*: In contrast to the previous mode, where the slave skill follows the master skill's progress, this mode does not assign a "master" nor "slave" role. Both skills have to converge together to a common progress state, in such a way that one skill adapts when the other is perturbed. This pattern is realized by imposing constraints on each skill's progress variables with:

$$c = \langle e = s_i, e_d = \{s_i^{\text{ff}}, s_j\}, C_{\text{proportional}} \rangle \quad (7)$$

where i and j are the indices of the two composed skills. The progress of skill- i evolves such that it follows its own feedforward term s_i^{ff} and react to the difference to its peer's progress s_j . This behavior is similar to the consensus algorithm presented in [15] applied to a simple two-agents scenario. The cooperating agents try to converge to a common information state - a progress variable in this case. This pattern operates on two skills:

$$\text{ComposePeers} : S1 \times S2 \rightarrow S3 \quad (8)$$

where the composed skills S1 and S2, and the resulting skill S3 all belong to Type-I.

B. Weight Scaling Pattern

The last composition pattern is intended for composing a progressing skill with a non-progressing counterpart. This pattern is useful for combining nominal with a reactive skill behaviors when their constraints are not compatible with each other's. For example, consider the composition of the Cartesian motion skill of Figure 2 with an admittance motion skill. We want a combined behavior such that the robot moves its end-effector along the path, but reacts whenever a human-induced force is applied by stopping its progress.

Combining the two skills by simply executing their constraints together does not result in the desired behavior. On one hand, the Cartesian motion skill imposes the following task constraint:

$$c_{\text{cartesian}} = \langle e = T_{\text{ee}}^{\text{world}}, e_d = T_{\text{path}}, C_{\text{proportional}} \rangle \quad (9)$$

where $T_{\text{ee}}^{\text{world}}$ is the robot's end-effector pose expression w.r.t. the world frame and T_{path} is the target pose. On the other hand, the admittance skill imposes:

$$c_{\text{admittance}} = \langle e = -K_a T_{\text{ee}}^{\text{ee}}, e_d = W_{\text{ee}}^{\text{ee}}, C_{\text{proportional}} \rangle \quad (10)$$

where K_a is the admittance matrix and $W_{\text{ee}}^{\text{ee}}$ is the measured wrench applied by the human. The admittance skill moves the robot end-effector with a scaled velocity proportional to the applied wrench. When the force is absent, the Cartesian motion skill enforces the robot to move along the path while the admittance skill enforces the robot to stop. These conflicting constraints result in an undesirable behavior.

This composition pattern addresses this problem by means of varying the skills' weights. In the nominal condition with no disturbance, the pattern assigns a higher weight to the progressing skill. Meanwhile, in the presence of disturbance, the reactive, non-progressing skill gets a higher weight. The weights allocation is done by following a smooth function to avoid unwanted jerky motion.

To this end, the weights are scaled with a sigmoid function that changes value according to an activation signal $\zeta \in \mathbb{R}$. In the case of Cartesian motion with admittance skill, the activation signal can be the norm of the measured wrench $\zeta = \|W_{\text{ee}}^{\text{ee}}\|$. In this paper, we choose a hyperbolic tangent function $w_{\text{scaling}} = \tanh(\alpha\zeta)$, where α is a scalar parameter to adjust the sigmoid's steepness. The skills' weight factors are then scaled accordingly, resulting in:

$$\bar{w}_{\text{composition}} = \begin{bmatrix} (1 - w_{\text{scaling}})S & 0 \\ \mathbf{0}_{1 \times 2} & w_{\text{scaling}} \end{bmatrix} \begin{bmatrix} \bar{w}_t^i \\ \bar{w}_t^j \end{bmatrix} \quad (11)$$

The superscripts i and j denotes the weights for the progressing and non-progressing skills, respectively. $S \in \mathbb{R}^{2 \times 2}$ is a diagonal, selection matrix whose elements can be set to '0' or '1' in order to select whether the scaling is applied to the task or progress constraints. This pattern can be expressed as an operation that is applied to two skills:

$$\text{ComposeWeightScaling} : S1 \times S2 \rightarrow S3 \quad (12)$$

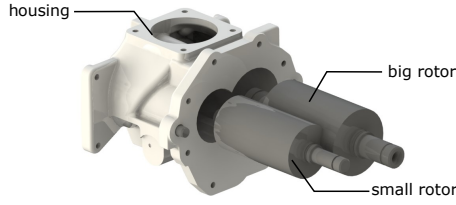


Fig. 4. The rotary compressor parts to be assembled. Note that the detail of the rotors' helical meshing is not visible.

where S1 can be either Type-I or Type-II skill, while S2 is a Type-III skill. Applying this operation results in S3 that inherits S1's type.

V. EXPERIMENTAL VALIDATION

This section reports the results of an assembly task performed by skills that were composed and executed with the proposed framework. The goal of the experiment is to show how the framework is applied to specify skills and their compositions in the presence of disturbances.

A. Problem scenario

We consider an assembly task of an industrial compressor which consists of two rotors as shown in Figure 4. The experiment is performed by a dual-arm setup consisting of a KUKA LBR iiwa and a LWR4 robots that are positioned in proximity. The two robots' workspace overlap such that collision can occur. Each robot assembles one rotor into the compressor housing and they both execute the task simultaneously in order to minimize idle time. In addition to collision between the robots, another non-nominal disturbance is the presence of human exerting force to the robot during execution time.

B. Implemented skills

1) *Cartesian Motion*: The Cartesian motion moves a designated end-effector frame from an initial to target pose. In order to obtain a smooth motion, a trapezoidal motion profile is defined to generate a Cartesian path as a function of a feature variable $T_{\text{path}} = f(\chi_{f_{pv}})$. T_{path} is equal to the target pose when $\chi_{f_{pv}} = t_{\text{end}}$, where t_{end} is the total duration of the motion profile. A progress constraint is imposed to $\chi_{f_{pv}}$ such that its value increases as time elapses. This skill belongs to the Type-I category, with a progress function that can be instantaneously evaluated and controlled.

2) *Force-Based Insertion*: Due to their tight clearance, the rotors assemblies are difficult to perform using a position-based skill. Therefore, they need to be assembled using a sensor-based skill that continuously checks the contact force. We specify the skill using the task constraints as described in [7]. This skill also belongs to the Type-I category.

3) *Collision Avoidance*: The collision avoidance skill is specified by imposing inequality constraints to the closest distance between two pairs of shape primitives. These shape primitives approximate the robot links and they are usually

modeled with simple geometry e.g. boxes or capsules to reduce computation cost. Due to its non-progressing behavior, this skill belongs to the Type-III.

4) *Admittance Motion*: Another Type-III skill that we implemented is a reactive skill that performs admittance behavior by reacting to an external force disturbance. In our experiment, this skill is used to reject the disturbance applied by a human.

C. Composing the Skills

By knowing the expected skill behaviors and their types, the application developers can compose them with the appropriate patterns. We developed the application by first creating a sequential composition of the nominal skills i.e. the skills that each robot executes without considering disturbances. The sequences are specified with the FSM model proposed in [16]. The second step is to compose the nominal skills with the reactive skills to address the human disturbance. To this end, weight scaling pattern is used for combining the admittance with the Cartesian motion behaviors.

The rotors need to be assembled one at a time to avoid collision. Therefore, when the two robots are about to collide, one needs to be prioritized over the other. In eTaSL, this can be realized by assigning different priorities or weights between the two robots' skills. Here, we opted for applying different weights such that the LBR iiwa has more priority than the LWR4 robot. Furthermore, for each progressing skills, larger weights are assigned to the task constraints relative to the progress constraints. This way, the robot stays close to its task space in the presence of disturbance or when the other robot pushes it back. The resulted composed skills is visualized as a simplified state diagram shown in Figure 6. The diagram shows the sequential transitions between skills as well as pairs of skills that are subject to composition patterns. In order to demonstrate the synchronization behavior, we impose a master-slave pattern to the pre-picking task of the LWR4 with the pre-insert task of the LBR iiwa. The experiment shows that the composed skills successfully perform the assembly tasks. The snapshots of the robots performing is shown in Figure 5.

D. Analyzing the Composition Behaviors

1) *Combination of the Cartesian Motion and Admittance Behaviors*: The weight scaling pattern is used in combining the nominal Cartesian motion and the admittance behavior during the pre-picking subtask of the small rotor. Here, the pattern uses norm of the measured wrench as the activation signal. This causes a change in the weights between the two skills, as shown in Figure 7 (normalized for readability). The disturbance were applied twice: between $t = 3.4$ to 6.2 and $t = 10.7$ to 12.3 seconds. Note that we did not compensate for robot's dynamic force, therefore the measured force values are not zero outside the disturbance period. As the force increases, the Cartesian motion skill's weight reduces while the admittance's weight increases. When the force is large enough, the robot gets pulled from its nominal progress, as shown in the bottom plot of Figure 7.

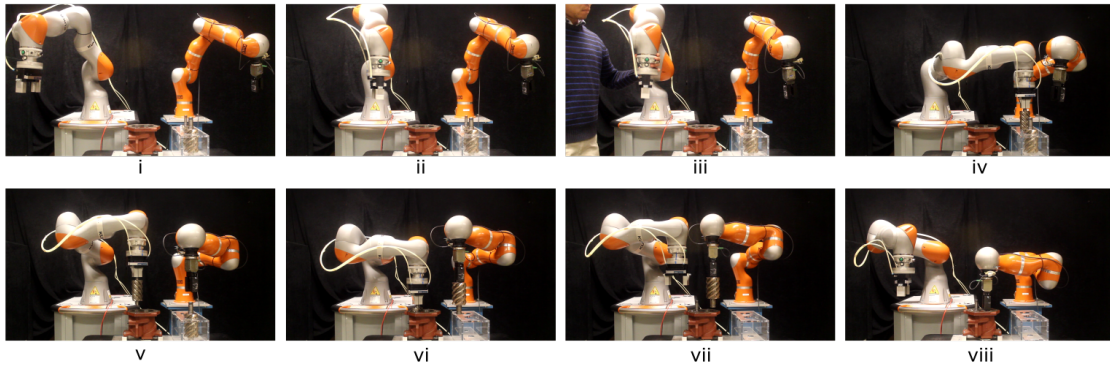


Fig. 5. Snapshots of the assembly process. Each robot performs a sequence of gripping and inserting the rotor. Human disturbance is acting on the LBR iiwa robot (iii).

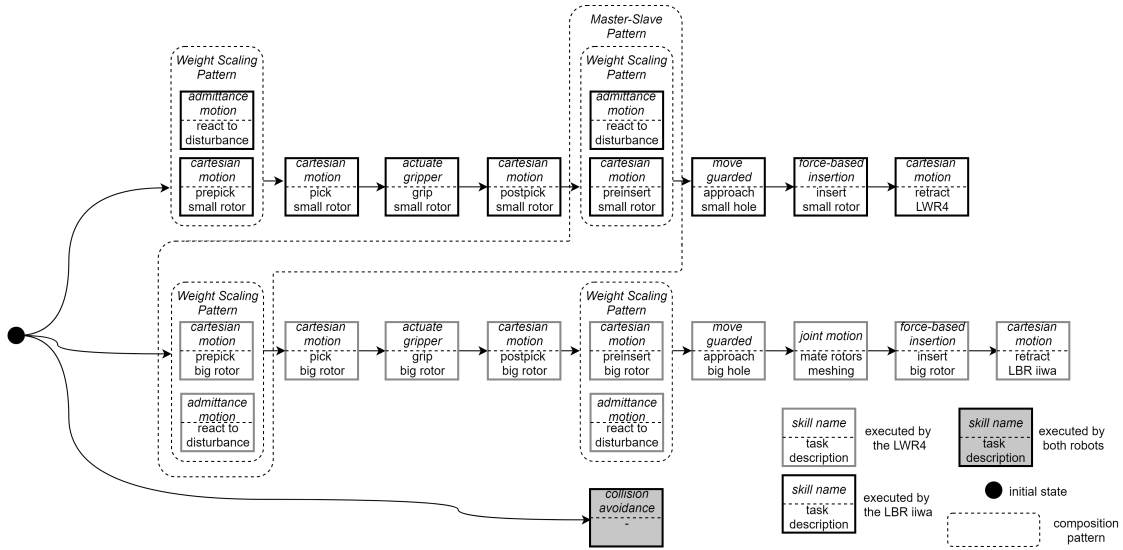


Fig. 6. The overall sequential and parallel compositions for the rotors assembly task. Description about each block is provided in the bottom right part of the figure. The arrows denote sequential skill transitions.

TABLE II
COMPARING THE FEATURE BETWEEN FRAMEWORKS

Feature	our framework	Pedersen [1]	Thomas [2]	Stenmark [3]	Nagele [4]
Skill implementation	eTaSL	TFF, motion planner	TFF	iTaSC, robot primitives	iTaSC
Continuous-level composition	parallel, coupled	n/a	n/a	n/a	parallel, independent
Discrete-level composition	FSM	sequencer	statechart	FSM, sequencer	statechart
Constraints resolution	priority, weight	n/a	n/a	n/a	priority
Task & progress constraints separation	yes	no	no	no	no
Domain-specific Language	no	no	yes	no	yes

2) *Synchronization Behavior of two Cartesian Motion Skills*: The pre-insert task of the small rotor is synchronized with the pre-pick task of the big rotor using the master-slave pattern (Figure 5.v). Here, the pre-pick task of LWR4 is assigned the "master" role while the pre-insert task of LBR iiwa is assigned the "slave" role. Their progress evolution is plotted in Figure 8. These two skills started at a different time. The pre-picking task is already active from the beginning of the experiment, but is unable to progress for some time (between $t = 13$ to 25 seconds) due to the more prioritized LBR iiwa obstructing its path. As soon as

it can proceed, the progress increases again and the LBR iiwa makes the transition from post-pick to pre-insert tasks ($t = 25.6$ seconds). Since the pre-insert task is subject to the master-slave synchronization, it quickly tracks the master's progress. The plot shows that the synchronization is achieved with small progress error.

VI. DISCUSSION AND CONCLUSION

This paper proposes a skill-based programming framework that enables application developers to compose reactive behaviors that handle disturbances and synchronization con-

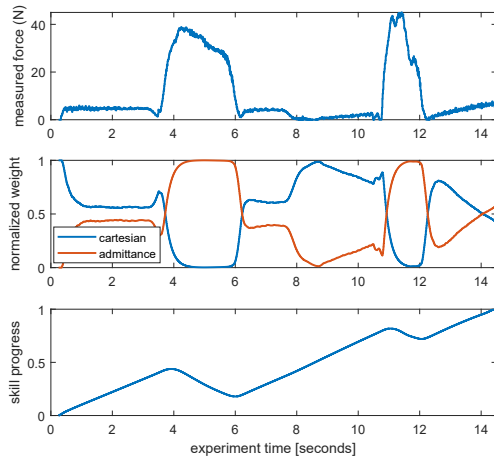


Fig. 7. The scaled skill weights during pre-picking task of the small rotor. The disturbance force and the skill progress are also shown.

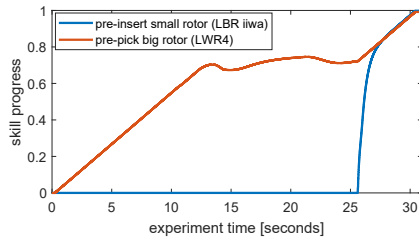


Fig. 8. The evolution of the skill progresses under master-slave synchronization pattern.

straints during the task execution. A skill specification that separates the task and progress constraints is modeled. Different skill types are classified based on how their progress is determined. Non-trivial composition of skills is performed by applying reusable composition patterns.

A comparison between our framework and the state of the arts [1] [2] [3] [4] is outlined in Table II. Our framework's main advantage is the higher composability in continuous level that allows to specify parallel and coupled behaviors. This composability is either not addressed or not formalized in [1] [2] [3]. Other frameworks are more flexible in the skill implementation. For example, [3] allows to use both iTaSC and robot-specific motion primitives to realize a skill. However, such flexibility also introduces problems when multiple skills are composed and executed together. When there are conflict between the skills, it is more difficult to resolve which skills should be prioritized. Due to its stronger ties with the underlying constraint-based task specification framework eTaSL, our work can use both priorities and weights to address this issue. As a topic for future work, we recommend the introduction of a domain-specific language (DSL) to support formal verification, as available in frameworks [2] and [4].

A test case of compressor rotors assembly performed by a dual-arm robot setup is evaluated to verify the effectiveness of the framework. The assembly task is subject to non-nominal conditions including the collision between the two

robots, perturbation by a human, and a synchronization constraint between selected skills. Desired deviating behavior is obtained by properly assigning different weights/priorities to the composed skills. The composition patterns facilitate combining nominal and reactive skill behaviors as well as achieving synchronization of the skills' progress. The experiment shows successful execution of the assembly task.

ACKNOWLEDGMENT

The work reported in this paper was supported by Flanders Make SBO PROUD and MULTIROB in Belgium.

REFERENCES

- [1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282–291, 2016. [Online]. Available: <http://dx.doi.org/10.1016/j.rcim.2015.04.002>
- [2] 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.
- [3] M. Stenmark and E. A. Topp, "From demonstrations to skills for high-level programming of industrial robots," *AAAI Fall Symposium - Technical Report*, vol. FS-16-01 -, pp. 75–78, 2016.
- [4] F. Nagele, L. Halt, P. Tenbrock, and A. Pott, "A prototype-based skill model for specifying robotic assembly tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.
- [5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," pp. 500–505, 1985.
- [6] O. Brock and O. Khatib, "Elastic Strips: A Framework for Motion Generation in Human Environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2004.
- [7] H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the "Task frame formalism" - A synthesis," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 581–589, 1996.
- [8] J. D. Schutter, T. D. Laet, J. Rutgeerts, W. Decre, R. Smits, E. Aertbelien, K. Claes, and H. Bruyninckx, "Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty," *The International Journal of Robotics Research*, vol. 23, no. 3, pp. 433–455, 2007.
- [9] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, "iTASC: a tool for multi-sensor integration in robot manipulation," in *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE, aug 2008, pp. 426–433.
- [10] E. Aertbelien and J. De Schutter, "ETaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *IEEE International Conference on Intelligent Robots and Systems*, 2014, pp. 1540–1546.
- [11] 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 *Advanced Robotics, 2009. ICAR 2009. International Conference on*, no. 8, 2009, pp. 1–6.
- [12] L. Halt, F. Nagele, P. Tenbrock, and A. Pott, "Intuitive constraint-based robot programming for robotic assembly tasks *," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.
- [13] M. R. Pedersen, L. Nalpantidis, A. Bobick, and V. Krüger, "On the Integration of Hardware-Abstracted Robot Skills for use in Industrial Scenarios," *2nd International IROS Workshop on Cognitive Robotics Systems: Replicating Human Actions and Activities*, 2013.
- [14] M. Stenmark, M. Haage, and E. A. Topp, "Simplified Programming of Re-usable Skills on a Safe Industrial Robot: Prototype and Evaluation," *ACM/IEEE International Conference on Human-Robot Interaction*, vol. Part F1271, pp. 463–472, 2017.
- [15] D. Kingston, Wei Ren, and R. Beard, "Consensus algorithms are input-to-state stable," pp. 1686–1690, 2005.
- [16] M. Klotzbücher and H. Bruyninckx, "Coordinating Robotic Tasks and Systems with rFSM Statecharts," *Journal of Software Engineering in Robotics*, vol. 1, no. January, pp. 28–56, 2012.