Combined Pose-Wrench and State Machine Representation for Modeling Robotic Assembly Skills

Arne Wahrburg, Stefan Zeiss, Björn Matthias, Jan Peters, and Hao Ding

***Abstract*— A new *Robotic Assembly Skill* (RAS) modeling framework is proposed. An assembly skill is a primitive that encapsulates the capabilities to coordinate, control and supervise an elementary robot task. To gain reusability of a primitive in alike robot tasks, the primitives are represented as generic templates that are parametrized for each situation with data from an assembly specification. A skill is represented in two ways, namely as a trajectory describing compliant motions in pose-wrench space and as a finite state machine. This approach comes with the potential to simplify robot programming and to improve robustness in robotic assembly due to inherent quality checking. The approach is implemented on an ABB YuMi robot performing the assembly of a programmable logic controller (PLC) I/O module.**

I. Introduction and Related Work

Currently, manufacturing increasingly moves from mass production towards mass customization [1]. This shift demands a new kind of flexible industrial robots, which must cope with uncertainties in order to operate in at least partly unstructured environments [1]. It has to be possible for non-experts to reprogram robots quickly on the shopfloor. These two goals compete as uncertainty handling requires sensor-based control strategies, which increases programming complexity compared to standard position control.

A possible solution to this conflict is the approach of *Assembly Skills*. The general idea is to store the ability to perform elementary robot actions in reusable primitives. A skill is such a primitive that allows the coordination, control and supervision of a specific task. The primitives can incorporate advanced task specifications, necessary control, and sensing capabilities, which potentially allows a skill to handle uncertainties during execution. As all of this information is encapsulated, the approach implies an emphasis shift from traditional position-based teaching to selecting and parameterizing predefined skill primitives to complete assemblies. Consequently, the skill primitives form a link between high-level planning and low-level control of task execution.

An early approach to use skill primitives as a representation of atomic robot motions in the context of assembly is given in [2]. Another skill-like system was used in [3] for robotic assembly using a rule-based logic combined with hidden Markov models. Skill primitives have been combined to discrete motion networks which represent tasks in [4]. The task frame formalism is employed to define manipulation primitives, which are used as building blocks for skill primitives. In [5], a constraint-based programming approach [6] is employed to create a skill-controlled robot motion system. Therein, the iTaSC framework [7] is used to interpret motion commands defined by constraint-based programming. A similar approach is presented in [8], where skills are represented by state machines and each state contains iTaSC-based motion commands. Additionally, a knowledge integration framework for storing, sharing, and reusing skills and other assembly knowledge was introduced in [9]. Discrete motion networks are also used in [10], where the networks represent force-over-position trajectories. Another approach is to represent skills by a dynamic system. In [11], spatially and temporally invariant dynamical systems are used. A canonical system is transformed to adapt to new environmental constraints. The approach is e.g. employed in [12] to acquire new skills by learning techniques. A novel approach of employing dynamic systems is presented in [13], where robot motions are described as flows in a simulated current of fluid. A recent implementation of a skill-based system is given in [14]. Therein, a new programming language based on UML/P statecharts is introduced to describe skills and the task frame formalism is used for motion description.

The main contribution of this paper is a new programming framework called *Robotic Assembly Skill* (RAS). Within this framework, robotic skills are represented using two components. First, a trajectory representation in pose-wrench space provides a general description for mating one part to another in a basic assembly action, comprising both positions/orientations and forces/torques. And, second, a

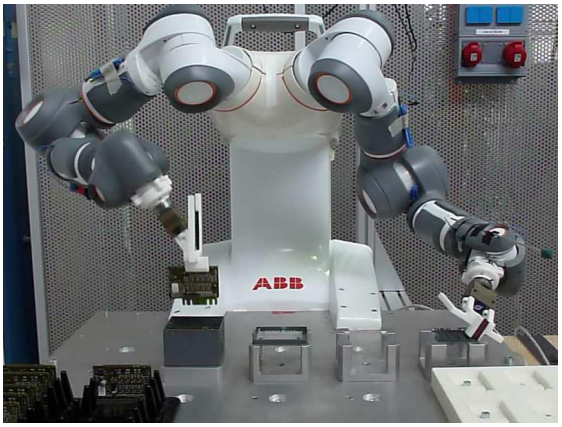


Fig. 1. ABB YuMi (`www.abb.com/yumi`), a dual-arm 7DOF manipulator, assembling a PLC I/O module employing the proposed RAS framework. The full assembly of the I/O module has been realized using the proposed approach. Two elementary operations, namely snap-fit assembly steps, are detailed in the paper to explain the proposed approach.

The authors are with the ABB Corporate Research Center, GERMANY {`firstname.lastname`}`@de.abb.com` and the TU Darmstadt, GERMANY `stefanzeiss.sz@gmail.com`, `mail@jan-peters.net`.

motion net representation coordinates such robot motions with environment interaction. The former principle has been introduced in a less generic form in [10]. The latter representation is also used in [5], [8], [14]. Combining both representations in a generic framework as proposed in this paper has two advantages. First, we can reuse generic representations by parameterizing skill templates. Second, the system may gain in robustness since the scheme is not purely position-based but forces and torques are also taken into account to supervise execution.

The paper is structured as follows. The RAS programming approach is sketched in Section II. Section III presents experimental results employing the RAS scheme to perform snap-fits with an ABB YuMi robot in the assembly of a PLC I/O module before a conclusion is given in Section IV.

II. The Robotic Assembly Skill Framework

To use robots in flexible production scenarios such as small part assembly, two main problems have to be solved. Firstly, robots need to be able to operate in partly unstructured environments under the presence of uncertainty. Secondly, robots need to be quickly adaptable to react to frequently changing production scenarios. A manipulator that can operate in the presence of uncertainty demands complex sensor-based control strategies. To adapt a robot quickly, it has to be possible to reprogram it without much effort, preferably by non-experts on the shopfloor. As the former requirement increases the programming complexity, the two goals compete. An approach to handle the two competing goals is the *Robotic Assembly Skill* framework depicted in Fig. 2. The idea is to establish an interface between high-level assembly planning and low-level robot controls, namely *skills*. In the following, the key elements of the scheme (application, task, skill, and motion) are briefly introduced before assembly skills are discussed in detail.

An *application* is defined to specify a complete product by a hierarchy of geometrical relations between sub-assemblies and individual components stored in a binary assembly tree. Furthermore, annotations of the tree nodes may contain information on how the parts have to be assembled and which resources are required. As an example application, consider the full assembly of a PLC I/O module pictured in Fig. 1.

A *task* is defined as a traversal step in the assembly tree. In this context, a task is limited to the scenario of two parts being put together. Attaching the module cover to the housing in the PLC I/O module assembly is an example of a task. Each task is instantiated by one or many skills which form a net of skills.

A *skill* is defined as an elemental manipulation constituting a step towards achieving a task. As an example, consider performing the actual snap-fit in the task of attaching the module cover to the housing. A skill contains all the necessary data to coordinate, control and supervise the actions necessary to fulfill this step. In the proposed RAS approach, a skill is represented by a generic template selected from a skill library and a specific instantiation is created from this template and data from the task. For different assemblies, the same generic template can be reused with tailored specific parameterizations.

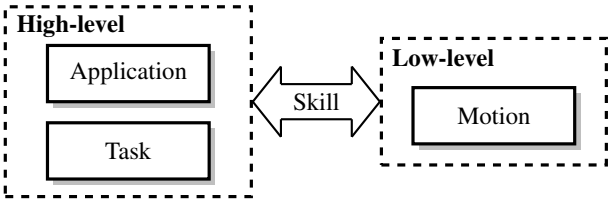


Fig. 2. Robotic Assembly Skills as an interface between high-level assembly planning and low-level robot controls – An application is understood to specify a complete product, with a task defined as putting together two parts forming a sub-assembly. A motion is defined as a constrained or unconstrained action, that can be position- or force-controlled. Skills can be regarded as an interface unburdening the user from directly programming low-level robot controls. Instead, reusable generic skill templates are combined to fulfill a task, where the individual skills are parameterized for the specific assembly at hand.

A *motion* is defined as a constrained or unconstrained action that describes which degrees of freedom of a reference frame are controlled in which way. The frame is associated with the parts to be assembled referring to the task frame formalism [4]. Specific frame and motion values are set using the parameters from the instantiated skill. In the example of performing the snap-fit as part of attaching the cover to the housing, the movement in negative z-direction until a specific force threshold is exceeded constitutes a motion. Instead of a motion, the data on this level can also represent an elementary tool action.

The robot program for the application should be easily and intuitively composed by selecting, combining and parameterizing appropriate skills from a library, according to the assembly tree specification. The effort for creating the library of generic skills is, however, to be shouldered by the robot supplier or system integrator. Their experts will have prepared separately the assembly skill library as part of the robot system software. The effort for creation of the low-level robot program code inside the skills is, therefore, no longer associated with the application user – a prospective advantage to present-day approaches.

A. Overview on Assembly Skills Representation

A skill is described by two components. The *motion net representation* describes the skill as a finite state machine, which is used for the discrete coordination and control of robot motions. Specifically, the motion net is used to switch between different continuous control actions. The *trajectory representation* represents the skill in pose-wrench space, and is used to supervise the continuous robot actions and to trigger transitions in the finite state machine.

A skill has input and output signals used for high-level supervision purposes. Based on these signals, the successor of the skill can be selected, for example. The signals are implicitly defined in a skill by states and transitions in its motion net representation, which are activated under certain conditions. Once the *precondition* is met, the execution of the related robot actions starts. A typical precondition would be that all involved parts are at a certain position.

The *completion signal* indicates that the last robot action relevant for the completion of a skill execution has been executed. After completion is indicated, the quality of the skill execution can be evaluated. The result of this evaluation is indicated by the *quality signal*. Several degrees of success can be represented, depending on how many different quality conditions are defined in a skill. Most commonly, only "success" and "failure" are distinguished. The execution of a skill is interrupted, if an *interruption signal* occurs. Based on this signal, predefined error recovery actions can be triggered.

In the following, both the trajectory representation and motion net representation will be detailed.

1) Trajectory Representation: The relative motion of two parts during assembly can be regarded as following a desired trajectory embedded in a 12D space spanned by the components of relative pose and wrench. The trajectory consists of a sequence of 12D state vectors describing the relative position, orientation, contact forces, and contact torques between two frames, each attached to one of the parts to be assembled. Pose and wrench usually refer to the motion of the end effector frame of the manipulator with respect to a reference frame in the workspace. Here, both frames are usually attached to parts either held by the end effector or located in a fixture in the workspace. One of the frames is called the *task frame*. In this frame, the pose-wrench configuration of the other part is expressed.

A trajectory expressed in 12D space does not explicitly depend on time. This property is useful as it allows the representation of a motion independent from time-related properties such as motion speed. It has to be noted, that not every motion needs all 12 dimensions to be specified for its control and supervision. In many cases, it is sufficient to consider a subset of these dimensions for supervision. As an example, consider the force-over-position trajectory of a snap-fit skill depicted in Fig. 3. In this case, the core characteristic of the skill is described in a 2D subspace spanned by position z and force F_z in the same direction. Intuitively, the trajectory can be partitioned into four segments, namely N_1 (free motion, no contact between latch and deflector), N_2 (latch increasingly bent by deflector), N_3 (latch sliding along deflector fully deflected), and N_4 (latch snapped in). A sketch of a typical snap-fit geometry can be found in Fig. 7 in the appendix where also a mathematical derivation of the trajectory is provided. It is to be noted that the if deflector length L_2 is very small, segments N_3 and N_4 can be merged, which is frequently the case in small parts assembly (cf. results presented in Section III).

We extend the approach to enable supervision of skill execution. To this end, the sequence of 12D state vectors describing the trajectory is augmented with another 12D vector specifying tolerance in each dimension. Therewith, a 12D hypertube is spanned. During execution, the actual pose-wrench value is compared to the hypertube. In case the actual value of pose-wrench is outside the specified hypertube, a transition in the motion net (cf. Section II-A.2) is triggered.

2) Motion Net Representation: The motion net contains the execution logic of the robot actions contained in a skill and is represented as a finite state machine. Tasks of the state machine are the coordination and control of robot motions. In the motion net, each state contains a robot action that is to be performed. To switch between robot actions, transitions are triggered. Accordingly, the motion net elements are referred to as motion net states M_i and motion net transitions T_{ij}. The motion net states contain control values, namely a controller setpoint c_i defining reference values in all task frame coordinates and a task frame. Furthermore, a trajectory segment N_k is associated to a motion net state for supervision.

As an example, consider the motion net representation of the snap-fit skill depicted in Fig. 4. From the initial state M_0 relating to segment N_1 in the trajectory (cf. Fig. 3), the transition T_{01} is triggered once the starting position from which the skill is to be executed is reached. State M_1 describes the robot motion in z-direction of the specified task frame. This motion is executed until the specified threshold in the force F_z is exceeded triggering transition T_{12}. Hence, M_1 and T_{12} relate to segment N_2 in the trajectory. Motion state M_2 continues movement in z-direction until the force

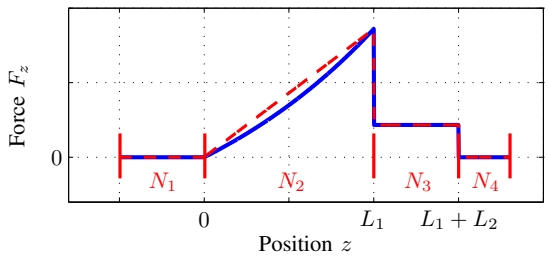


Fig. 3. Force-over-position profile of a snap-fit – The blue line shows the Force F_z that has to be applied in order to move the latch in z-direction as a function of position z, determined from equation (3) in the appendix. The dashed red line shows a piecewise linear approximation valid for $L_1 \ll l$, i.e. guidance length being small compared to latch length. This condition is typically met for latches in small parts assembly. The trajectory is partitioned into four characteristic segments (N_1 latch unbent, N_2 latch bending, N_3 latch sliding fully deflected, N_4 latch snapped in). The dimensions L_1, L_2, and l refer to Fig. 7 in the appendix.

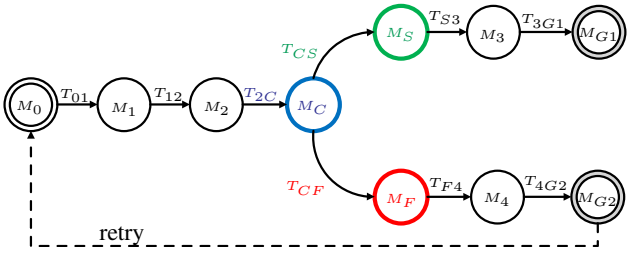


Fig. 4. Motion net representation of the snap-fit skill – From the initial state M_0, the transition T_{01} is triggered to start the execution of the skill. The states M_1 and M_2 resemble a movement in z-direction until a specified force threshold is exceeded (T_{12}) and fallen below of again (T_{2C}), taking the motion net to completion state M_C. Evaluation of contact forces after completion of execution triggers either T_{CS} in case of successful execution or T_{CF} to flag an error, respectively. In the latter case, a simple retry is triggered. This scheme could be extended towards advanced error handling routines. The states M_3 and M_4 resemble motions taking the manipulator back to the pre-snap pose, and $M_{G1/2}$ mark the final states.

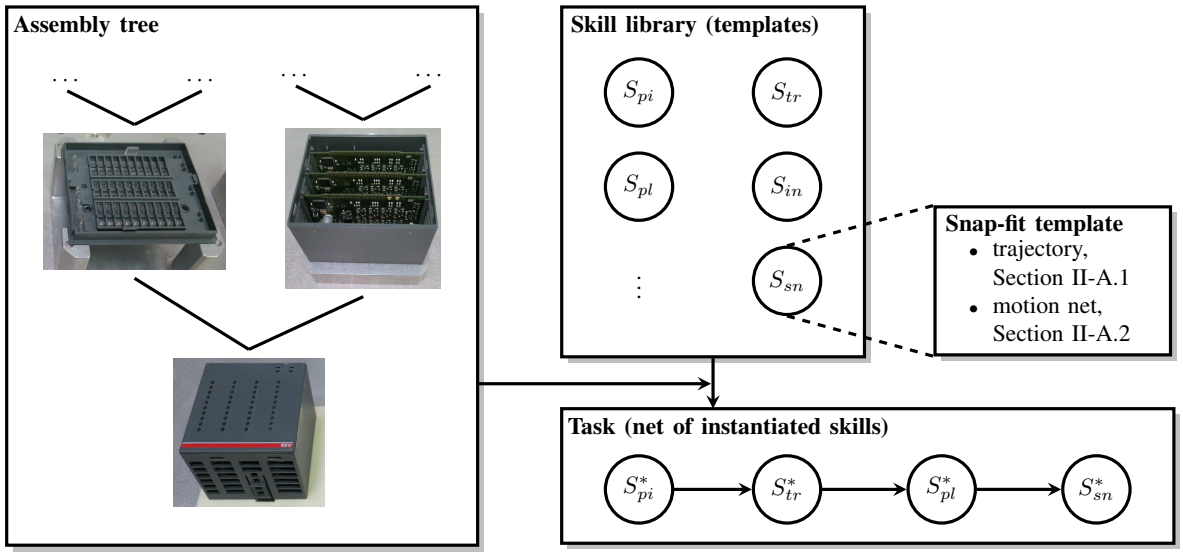


Fig. 5. Skill selection from skill library and parametrization based on assembly tree – In the example, the skill library contains S_{pi} (pick up), S_{pl} (place), S_{tr} (transfer), S_{in} (insertion), and S_{sn} (snap-fit). The skill library can be extended to include further skills. Each skill has two core components, a trajectory representation to supervise motions within a skill and a motion net representation to coordinate motions in the skill. The assembly tree shown in the figure is a snapshot of the full PLC I/O module assembly. Assembling the housing and cover is a task, merging two sub-assemblies into the final product. The task is a net of instantiated (i.e. parameterized) skills S^*_{pi}, S^*_{tr}, S^*_{pl}, and S^*_{sn}. The skills are selected from the library and parameterized either according to annotations in the assembly tree or by manual parameterization by the user.

F_z degrades below the specified threshold (transition T_{2C}) relating to segment N_3. The state M_C marks completion of the skill execution. The remaining part of the motion net is devoted to determining whether the skill has been executed successfully. If the force remains below a specified threshold in segment N_4 of the trajectory, transition T_{CS} is triggered taking the motion net to the success state M_S. Transition T_{S3} is always true, so the state M_3 is activated. The purpose of M_3 is to move the robot back to the starting position of the skill. Once the starting position has been reached, transition T_{3G1} is triggered taking the motion net to the final state M_{G1} and the completion signal is sent. If the force in segment N_4 is above a specified threshold, it can be inferred that the skill has not been executed successfully. In this case, transition T_{CF} is triggered to activate the state M_F. Similar as for a successful execution, transition T_{F4} is always true activating state M_4 taking the robot back to the starting position. Upon reaching this position, transition T_{4G2} is triggered activating the final state M_{G2}. Since this state marks a failed execution of the snap-fit skill, a retry or other error handling strategies can optionally be triggered.

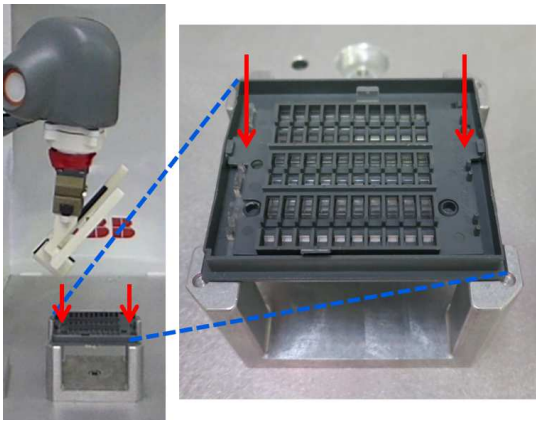
B. Skill Selection and Parameterization

As described in Section II-A, parameterized skill primitives are connected into nets to fulfill assembly tasks. Reusability is a key property of such a package, as the idea is to have a limited set of skill primitives that are capable of fulfilling a wide variety of complex tasks by connecting them. To this end, skill primitives are stored in a skill library as generic templates that can be reused in alike situations (s. Fig. 5). Each template skill is contains the 12D trajectory representation and the motion net representation as elaborated in Section II-A. To represent the task, the skills are selected from the skill library, connected to form a net of skills, and adapted to the specifics of the task at hand by setting a limited set of parameters. The specific values of the parameters may depend on the geometry of the parts to be assembled, their material, etc. Therefore, a specific skill parametrization is created for each situation.

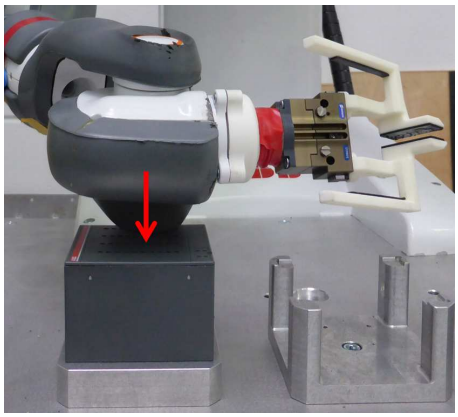
For the snap-fits in the PLC I/O module example, three parameters are mandatory. First, a task frame has to be specified which in the example coincides with the base coordinate frame. Second, a pre-snap position has to be specified, marking the TCP (tool-center-point) pose to start the snap-fit execution from. Third, a force threshold has to be set. Exceeding this threshold will trigger transition T_{12} and T_{2C}, respectively, in Fig. 4. Optionally, another threshold can be specified to evaluate contact forces after completion of the skill to assess the execution quality (i.e. "success" or "failure"). For the user, the workload shifts from low-level robot programming to the task of selecting appropriate skills from the library and parameterizing them.

C. Skill-based Assembly Execution

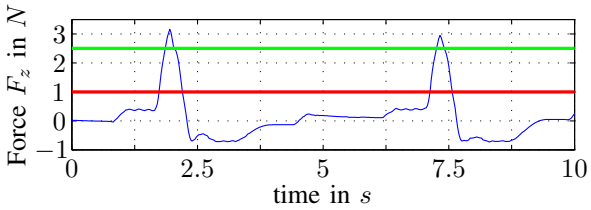
Once all tasks have been set up by creating nets of instantiated skills, they are executed in the order defined by the assembly tree. The finite state machines on task and skill level are employed to coordinate the assembly. From the states in the motion net representation of a skill, low-level robot commands are generated. In the snap-fit example, all these commands are force-supervised, position-controlled movements, but the RAS framework also allows for force-controlled actions. The execution of the movements is supervised by comparing actual pose and wrench data to the trajectory defined in the parameterized skill and trigger corresponding transitions in the motion net. For supervision of contact forces, either force/torque sensing or contact force estimation [8], [15], [16], [17] can be employed. While



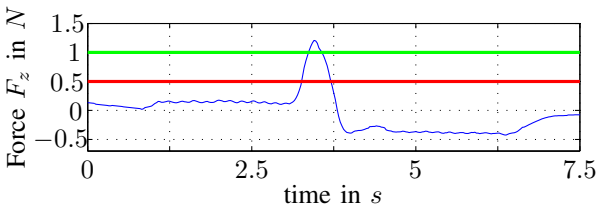
(a) First snap-fit example – The light guide cover is attached to the module cover by performing two snap-fits, one at each of the positions marked by the red arrows.



(b) Second snap-fit example – The cover is attached to the housing by executing a snap-fit skill in z-direction with the padded robot wrist at the position marked by the red arrow.



(c) Estimated contact force F_z for the light-guide-to-cover snap-fits depicted in Fig. 6(a).



(d) Estimated contact force F_z for the cover-to-housing snap-fit depicted in Fig. 6(b).

Fig. 6. Snap-fit scenarios in the PLC I/O module assembly and corresponding evaluation of estimated contact forces in z-direction over time – The green line resembles the threshold that has to be exceeded to mark completion of the skill execution. The red line represents the threshold that is evaluated to determine whether the execution has been completed successfully. In case the estimated contact force decreases below the red line after exceeding the green threshold, completion is marked successful. Notice the different parameterizations of the skill reflecting the different geometries and materials of the parts involved.

supervision enables a quality assessment of the execution, it can also be regarded as the basis for systematic error handling routines.

III. Experimental Results

The proposed RAS framework has been applied to perform a complete assembly of a PLC I/O module using an ABB YuMi robot (formerly known as ABB Dual Arm Concept Robot [18], [19]). The application involves pick, place, transfer, insertion, and snap-fit skills. The two types of snap-fit operations involved in the assembly are depicted in Fig. 6. The snap-fits visualized in Fig. 6(a) are performed with an edge of the gripper attached to the robot wrist. Their purpose is to attach a light guide cover to the module cover. The snap-fit visualized in Fig. 6(b) completes the assembly by mating module housing and cover.

Using a traditional position-based approach, three positions would have to be taught for each snap-fit. A pre-snap position from which to start the movement in z-direction, the actual snap position resulting in the latch snapping in and a post-snap position to retract to after the snap-fit. Especially the actual snap position is tedious to teach since it requires high precision in a purely position-based approach. If the target position is too high in z-direction, the snap-fit will not be executed successfully while a too low z-position might cause parts to break. With the proposed RAS approach, only one position has to be taught, namely the pre-snap position. Afterwards, the force thresholds have to be set. Furthermore, by evaluating contact forces, the RAS approach allows to detect whether the operation has been successfully completed which is not possible in purely position-based schemes.

Due to different geometries and material characteristics of the parts involved, the trajectories of the snap-fits shown in Fig. 6(a) and Fig. 6(b) are expected to differ. However, the general shape is preserved and the same skill template can be reused by creating different instantiations for the respective scenarios. This thought is verified by Fig. 6(c) and Fig. 6(d), respectively, showing estimated contact forces during execution of the snap-fits. The contact forces were estimated using the scheme proposed in [17] and we emphasize that only joint angle and motor torque information is needed. No additional sensing (e.g. joint torque or external force/torque sensors) has been used. While the variations in the range of $\pm 0.5N$ are due to measurement and process noise, the two spikes exceeding the force threshold of $2.5N$ in Fig. 6(c) mark the completion of the two snap-fit operations in the first example. For the second example, the threshold to detect completion of the snap-fit is set to $1N$ as shown in Fig. 6(d). Since the contact force falls below the specified threshold of $1N$ for the first example ($0.5N$ for the second example), successful completion of execution can be concluded.

IV. Conclusion

A skill-based framework for robot programming has been proposed. The approach relieves the user of low-level robot programming by providing reusable templates for robotic skills. For different applications and tasks, these templates have to be properly parametrized. The main benefits of the approach are potentially reduced teaching time, simplified robot programming and gain in robustness in robotic assembly. The framework has been successfully implemented to perform a full PLC I/O module assembly using an ABB YuMi robot. Future work will be devoted to the acquisition of new robotic skills and to assisting the user by providing a natural and intuitive interface for skill parameterization in an industrial environment. Systematic error handling strategies are also to be investigated.

Appendix – Modeling Snap-Fit Forces

Mathematics of snap-fits are studied in detail in [20], [21]. To derive a trajectory representation of a snap-fit, it is assumed that the task frame of the motion is set in such a way that the predominant direction of the motion is z and the latch is bent in perpendicular x-direction (cf. geometry sketched in Fig. 7). Assuming a diagonal inertia tensor, the deflection x can be found as a function of the deflecting force F_x as $x = l^3 F_x/(3EI)$, where l is the length of the latch, E is its modulus of elasticity, and I the area moment of inertia. Assuming a friction model with friction force F_R related to normal force F_N by $F_R = \mu F_N$, the friction force is employed to relate forces F_z and F_x as

$$F_z(z) = F_x \underbrace{\frac{\mu + \tan\left(\alpha_{\text{eff}}(z)\right)}{1 - \mu \tan\left(\alpha_{\text{eff}}(z)\right)}}_{\mu_{\text{eff}}(z)}, \; 0 \le z \le L_1. \tag{1}$$

It has to be noted that the effective latch guidance angle increases due to bending of the beam according to

$$\alpha_{\text{eff}}(z) = \alpha + \arctan\left(\frac{z\tan(\alpha)}{l + L_1 - z}\right), \; 0 \le z \le L_1. \tag{2}$$

With the effective latch guidance angle, the force-over-position profile of a snap-fit is given by

$$F_z(z) = \begin{cases} 3\mu_{\text{eff}}(z) EI \tan(\alpha) \frac{z}{(l+L_1-z)^3}, & 0 < z \le L_1, \\ 3\mu EI \tan(\alpha) \frac{L_1}{l^3}, & L_1 < z \le L_1 + L_2, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

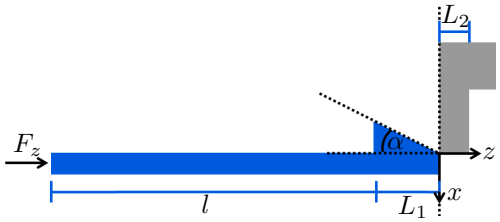


Fig. 7. Geometry of a snap-fit – Latch with total length $l + L_1$, guidance length L_1, deflector length L_2, and guidance angle α is moved in z-direction and deflected in x-direction by driving force F_z.

References

[1] S. Bøgh, O. Nielsen, M. Pedersen, V. Krüger, and O. Madsen, "Does your robot have skills?," in *Proc. of International Symposium on Robotics*, 2012.

[2] T. Hasegawa, T. Suehiro, and K. Takase, "A model-based manipulation system with skill-based execution," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 535–544, 1992.

[3] B. J. McCarragher, G. Hovland, P. Sikka, P. Aigner, and D. Austin, "Hybrid dynamic modeling and control of constrained manipulation systems," *IEEE Robotics & Automation Magazine*, vol. 4, no. 2, pp. 27–44, 1997.

[4] T. Kröger, B. Finkemeyer, U. Thomas, and F. M. Wahl, "Compliant motion programming: The task frame formalism revisited," in *Proc. of Mechatronics and Robotics*, 2004.

[5] R. Smits, *Robot Skills: Design of a Constraint-Based Methodology and Software Support*. PhD thesis, KU Leuven, 2010.

[6] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decre, R. Smits, E. Aertbelien, K. Clase, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *International Journal of Robotics Research*, vol. 26, pp. 433–455, 2007.

[7] R. Smits, T. D. Laet, K. Claes, H. Bruyninckx, and J. D. Schutter, "iTASC: A tool for multi-sensor integration in robot manipulation," in *Proc. of Multisensor Fusion and Integration for Intelligent Systems*, 2009.

[8] M. Linderoth, *On Robotic Work-Space Sensing and Control*. PhD thesis, Lund University, 2013.

[9] M. Stenmark and A. Stolt, "A system for high-level task specification using complex sensor-based skills," in *Proc. of Robotics: Science and Systems*, 2013.

[10] T. Nagai and S. Aramaki, "The representation method of robotic assembly task with click action," in *Proc. of Int. Symposium on Power Electronics, Electrical Drives, Automation and Motion*, 2008.

[11] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Proc. of Neural Information Processing Systems*, 2002.

[12] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer, "Robot skill learning," in *Proc. of European Conference on Artificial Intelligence*, 2012.

[13] H. G. Mayer, I. Nagy, A. Knoll, E. U. Braun, R. Lange, and R. Bauernschmitt, "Adaptive control for human-robot skill transfer: Trajectory planning based on fluid dynamics," in *Proc. of IEEE International Conference on Robotics and Automation*, 2007.

[14] U. Thomas, G. Hirzinger, B. Rumpe, and C. Schulze, "A new skill based robot programming language using UML/P statecharts," in *Proc. of IEEE International Conference on Robotics and Automation*, 2013.

[15] D. P. Le, J. Choi, and S. Kang, "External force estimation using joint torque sensors and its application to impedance control of a robot manipulator," in *Proc. of International Conference on Control, Automation and Systems*, 2013.

[16] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson, "Force controlled robotic assembly without a force sensor," in *Proc. of IEEE International Conference on Robotics and Automation*, 2012.

[17] A. Wahrburg, S. Zeiss, B. Matthias, and H. Ding, "Contact force estimation for robotic assembly using motor torques," in *Proc. of IEEE International Conference on Automation Science and Engineering*, 2014.

[18] H. Ding and B. Matthias, "Safe human-robot collaboration combines expertise and precision in manufacturing – a paradigm for industrial assembly in mixed environments," *atp-edition*, vol. 10, pp. 22–25, 2013.

[19] S. Kock, T. Vittor, B. Matthias, H. Jerregard, M. Kallman, I. Lundberg, R. Mellander, and M. Hedelind, "Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot," in *Proc. of IEEE International Symposium on Assembly and Manufacturing*, 2011.

[20] P. R. Bonenberger, *The First Snap-Fit Handbook*. Hanser, 2005.

[21] R. W. Messler Jr., *Integral Mechanical Attachment*. Elsevier, 2006.