A System Architecture for Constraint-Based Robotic Assembly with CAD Information

Mathias Hauan Arbo*, Yudha Pane†, Erwin Aertbeliën† and Wilm Decré†
*Department of Engineering Cybernetics
NTNU, Norwegian University of Science and Technology
†Robotics Research Group, core lab Flanders Make
Department of Mechanical Engineering, KU Leuven

***Abstract*—A system architecture is presented to generate sensor-controlled robot tasks from knowledge encoded in a CAD model. This architecture consists of an application layer where the user annotates assembly tasks in the CAD software. A process layer infers the specific robot skills and parameters from the CAD model and annotated data. A control layer executes the complex, force-controlled tasks. A proof-of-concept implementation is made, consisting of an application layer implemented in FreeCAD and a process layer that focuses on using fuzzy inference to generate appropriate skill-dependent process parameters from the geometric CAD information and annotations in the CAD model. In the control layer, a constraint-based control framework is used to robustly execute the assembly tasks. The system is validated on a challenging assembly task involving the assembly of screw compressor parts.**

I. Introduction

Manufacturing automation is entering a new era, where a high degree of customization of the product is expected by the client to which manufacturers must adapt quickly. This era is associated with human-robot collaborative workcells, short time to deploy, and tighter coupling of design and manufacturing.

Computer-Aided Design (CAD) software is a rich source of information for robotic assembly processes that may benefit a broad range of companies. The information includes part dimensions, geometric features, contact situations, etc. An early constraint-based description of pose relations between geometric features on assembly parts was presented by Ambler and Popplestone [1]. Reference frames attached to the geometric features were used to define the end-product of assembly in terms of "fits" or "against" relationships. The relative pose between these frames was described by equality or inequality constraints.

Autopass [2] is an early CAD-based assembly programming system that starts from a workpiece rather than a robot-centric perspective. Archimedes 2 [3] describes an architecture where the CAD model and assembly description, sequence planning, and skill execution are separated into modules. One of the most advanced systems is HighLap [4] where the CAD model is annotated with a small set of simple semantic assembly descriptions. The descriptions give constraints similar to Ambler and Popplestone that are used to find remaining degrees-of-freedom. HighLap uses the task frame formalism to execute force-controlled motion [5]. Neto et al. argued that integration of robot path programming in CAD software would benefit small and medium-sized enterprises where programming costs hinder automation [6]. A more recent approach by Perzylo et al. uses object-centric programming based on geometric constraints between parts to simplify the robot motion programming [7]. The comparison of the object-centric programming and classical teach-pendant based programming shows that object-centric programming is faster.

The robot control layer of a CAD-based assembly system can simplify or complicate the architecture. We advocate for: a workpiece perspective, composability, and sensor integration. A workpiece perspective allows us to specify control relative to the parts. Composability allows us to combine and include aspects of the environment, workpiece, and robot. To support a larger class of assembly situations, force control and easy sensor integration are key. Mason's task frame formalism [8] presents an early force and position control in the task domain. It shows how different control modes can be applied independently along an instantaneous task frame's directions, both in translation and rotation.

Motions are easily defined w.r.t. the workpiece, environment or robot using constraints, and the constraints are naturally composable. In [9], De Schutter et al. describe a procedure to design a robot controller that can deal with sensor interactions and motion in contact, using a set of auxiliary frames to systematically describe the constraints. This approach gave rise to the iTaSC software framework [10], a systematic approach for constraint-based programming that allows us to specify complex sensor-based skills. Subsequently, expression graphs are used to simplify constraint-based programming leading to a specification language eTaSL [11]. eTaSL scripts are used to specify the continuous behavior of the controller using constraints that relate to geometry or sensor-input. eTaSL scripts can specify monitors that trigger events into a restricted finite state machine (rFSM) [12]. This rFSM describes the discrete switching between different continuous control actions. A control layer using eTaSL/eTC and rFSM can perform assembly tasks in dynamic and uncertain environments, but requires specific process parameters to be defined, such as the force magnitude

during assembly or the dither amplitude during insertion. It is difficult to completely determine appropriate values for these parameters in a model-based way, so they are typically tuned. In this paper we want to facilitate the generation of assembly skills from CAD data by automatically determining process parameters using a fuzzy inference module.

The key contribution of this paper is the design of an architecture and a prototype implementation that allows us to generate controllers for assembly tasks requiring complex sensor-based interaction. This is done by splitting the required parameters for these assembly skills into two groups. One group of application and geometry-related parameters is generated from CAD model information. Another group of process parameters is determined using a fuzzy inference module. The resulting parameters are then used to generate a skill specification. These specification files can then be used to execute the specified skill on a reactive constraint-based robot controller. Using this architecture, complex force-controlled assembly skills can be executed even though the skills rely on empirical parameters.

Section II describes the system architecture and its basic concepts such as skills and tasks, and the inference module. Section III describes the software and hardware of the experimental setup, CAD workbench, the implemented skills and parameter inference. The experimental results are presented in Section IV. Section V discusses the experimental results.

II. System Architecture

The system architecture is outlined in Fig.1 and is split up into three layers: the application layer, the process layer, and the control layer. In the *application layer* we have a workpiece-centric view, where the user annotates the CAD model with the assembly tasks. In the *process layer*, assembly is considered from the point of view of the robotcell, and a planning and inference system ensures appropriate selection and composition of robot skills. The skills involved have application parameters and process parameters. Application parameters can be extracted from the CAD data, e.g. feature frame, or insertion length. Process parameters relate to the CAD data but without a clear underlying model. They may be empirical or have a range of values that produce acceptable results e.g. insertion force, or amplitude of anti-jamming dither. The inference module generates the value of these parameters. In the *control layer* the appropriate eTaSL skills are loaded together with the application parameters and the process parameters. A finite-state machine handles execution of the discrete states.

A. Skill and Task

Skill and task are often used interchangeably in the robotics literature. In this paper the terms refer to two different concepts.

A *task* is a piece of work to be undertaken, a *skill* is a particular ability. Assembly tasks are high-level assembly specifications in the application layer, and skills are related to particular actions the robot can perform. We differentiate

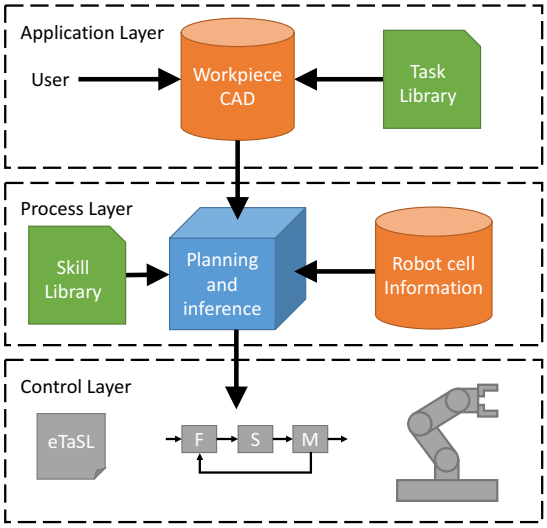

Fig. 1. System architecture.

between atomic skills and composed skills. The skills are defined by a set of specification files.

Atomic skills are eTaSL scripts that have: a configuration, inputs, outputs, and event specifications. The configuration is a set of parameters that are prepended to the eTaSL script and are constant during skill execution. These could be application parameters coming from the CAD model or process parameters generated by the inference module. The inputs bring information from continuous sources such as sensors. The outputs are mainly used for logging or analysis purposes. Monitors can trigger events that denote success or failure modes of the skill.

A common approach in robotics is to have states in finite state machines denote motion goals, e.g. *move_to_grasp_location*. This puts the skill complexity in terms of the states. An approach more often found in computer games is to have states denote operational modes, e.g. *walk*, *run*, or *move_cartesian*, and then redefine the parameters of these states during runtime. This reduces the states used and puts the complexity in the transitions. We have used the operational modes approach as it is easy to reuse a finite state machine with new application parameters, and as transition events denote the momentary situation that the robot is in. A *composed skill* is therefore an rFSM where each state corresponds to an atomic skill to be executed, and a function call associated with each transition event. The function call invokes external services such as grippers and tools, and loads the configuration parameters of the next atomic skill to be executed.

From a workpiece perspective, inserting a peg into a hole with large or small clearance is the same task, but they may need completely different insertion control strategies, e.g. when there is a large clearance a pure position-based approach could be sufficient, while small clearance would

necessitate force-control strategies. Therefore we map a task to a set of potentially applicable composed skills.

Going from task to composed skill is done by first finding the geometric primitives involved, and then using the CAD data describing these primitives to select the composed skill from a set of composed skills associated with a task on such geometric primitives, see Fig.2. In conclusion, tasks are generalisable and workpiece-centric, while skills correspond to an ability of the robot system to realize a task in a specific way.

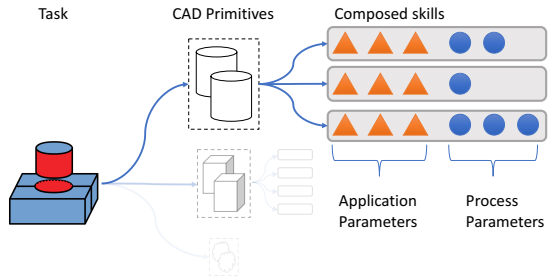

Fig. 2. A task is annotated between two geometric primitives. This task for these geometric primitive types can be performed by a set of composed skills. The CAD data of the geometric primitives is used to select the composed skill, which has application parameters and process parameters.

B. Planning and Inference Module

The planning and inference module has three main purposes: plan the assembly sequence, compose skills based on the task and the workcell, and generate the appropriate configuration parameters. Optimal planning of assembly involves optimal task sequencing, optimal trajectory planning, and selecting the optimal parameter values. These domains are interconnected, but they are assumed separable in the scope of this article. This article does not give a planning strategy but outlines how the task annotations tie in to the literature on assembly sequences.

1) Assembly Sequence Planning: tasks and parts form a *liaison graph*. In liaison graphs, a node represents a part, and an edge represents an assembly situation [13]. A *precedence graph* specifies which edges in the liaison graph should be completed before others. This precedence graph represents all feasible assembly sequences. In [14], Homem De Mello and Sanderson show the relation between precedence graphs and other assembly representations such as And/Or graphs. The precedence graph is created in the application layer. Assembly of geometric features gives a defined assembly direction and virtual disassembly in the CAD software along these lines provides a suggested precedence graph. The precedence graph is passed from the application layer to the planner for generation of the assembly sequence. Moving to grasp, changing tool, reorientation, and other workcell-related composed skills are added to the assembly sequence by the planner. Robot cell information such as which parts and tools are available, or are collaborative workspace skills required, is an essential plan of the planning module. This information must be stored in a robot cell database describing the different setups available for the manufacturer. As the underlying control layer is robot-agnostic and the skills are transferrable, we view this as a problem to be addressed in the planning module.

2) The Parameter Inference Module: given the geometric information between two mating parts and the selected skill, the inference module generates the appropriate process parameters to ensure successful assembly. Here we present an inference module for the insertion task of a cylindrical object in a tight-tolerance situation. This task would be extremely difficult to perform with only position control, hence the necessity of a force-controlled skill. We generalize the task as a peg-in-hole problem.

A number of process parameters need to be tuned for optimal insertion behavior. From existing literature [15], it is known that the insertion behavior is determined by a number of factors such as the peg's dimension and the peg-hole clearance. Even for a peg-in-hole assembly with relatively simple geometry, accurate process modeling is difficult to achieve for narrow clearances. Due to hyperstatic contact situations, not all contact forces are always externally observable. We therefore use a data-driven approach to determine the process parameters. To infer parameters in an uncertain or stochastic situation, a fuzzy inference approach is chosen [16]. Our fuzzy inference module uses the peg's length, peg's diameter and clearance to estimate the appropriate insertion force and dither amplitude.

III. Implementation

A. Assembly Use Case

We consider an assembly of a large and a small rotary-screw compressor (see Fig.3). The large compressor is composed of >30 parts and the small compressor is composed of >15 parts. To limit the scope of this demonstration, we focus on the assembly of the rotors and the housing lid. The housing is attached to a fixture and all insertions are in the same direction. Each compressor has a small and large meshing helical screw rotors. The top of the rotors go through the housing lid trough two chamfered holes. All parts to be assembled are placed in known poses. Fig.8 shows the resulting assembly sequence.

B. Application Layer

The application layer is implemented as a workbench in FreeCAD 0.16 [17]. FreeCAD is an open-source parametric CAD program. We implemented three task classes: *Insert*, *Place*, and *Screw*. Each of them is associated with two faces, and two *FeatureFrame* objects. To transfer information about the geometric features and their location, we implemented a tool for creating reference frames on geometric primitives. *FeatureFrame* objects can be instantiated on a selected vertex, edge, or face. If the selected geometric primitive has a center, center of mass, axis, or focus, the feature frame can be placed at the attribute with z-axis oriented along the axis if possible. On edges the frame can be placed along the edge

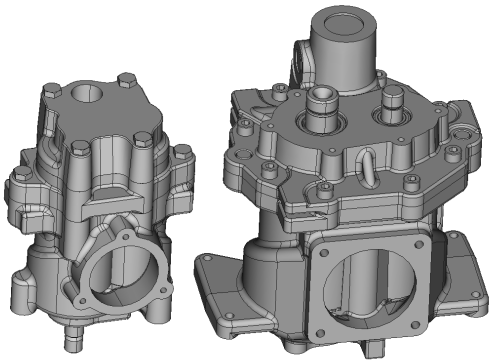

Fig. 3. CAD model of the two compressors side by side.

with x-axis aligned with the edge tangent. On the face the frame is placed with z-axis aligned with the normal. The parts are exported as STL meshes for visualization, and a JSON file describing the feature frames and the attributes of the geometric primitives relative to the mesh origin. The feature frame part of the workbench is publicly available [18]. By creating task instances between parts we form a liaison diagram, see Fig.4. During instantiation of an *Insert* or *Screw* object, we also instantiate a *FeatureFrame* object denoting the instantaneous task frame for force control. Since assembly sequence planning is not the focus of this paper, the assembly sequence was manually determined while specifying the task. The task instances, in order of execution, are: *insert_littlerotor*, *insert_bigrotor*, and *place_lid*. The tasks are exported in JSON files with reference to the part names and feature frame names. Grasp location is assumed to be known and is annotated as a *FeatureFrame* instance on the part.

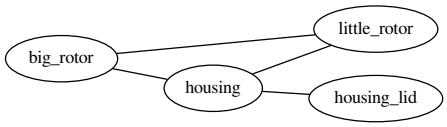

Fig. 4. Liaison diagram for the assembly use case. Although the rotors are in contact with the lid, the chamfers allow us to use a place task defined between the housing and the lid.

C. Implemented Skills

In this use-case each task maps to a single composed skill. When the skill set of the system grows this will no longer be the case. The two implemented composed skills are composed of a larger number of atomic skills, as listed in Table I. As the table shows, the *insert_littlerotor* instance uses *grasp_and_insert* as the suitable composed skill. This composed skill consists of three different atomic skills: *move_cartesian*, *guarded_cartesian*, and *cylinder_insert*. The transition between these atomic skills is shown in Fig. 5. When *e_start* occurs, we set the goal of *move_cartesian* to reach a position offset a constant distance along the z-axis of the grasp frame. Success in reaching the goal is associated with the *e_pregrasp* event. When *e_pregrasp* occurs the goal of *move_cartesian* is set to the grasp frame with success of the atomic skill being associated with *e_grasp*. When *e_grasp* occurs we have reached the grasp pose and engage the gripper and set the next goal to a z-axis offset from the grasp frame that we call post-grasp, and success gives the *e_postgrasp* event. The procedure is repeated for the z-axis pre-hole locations. When *e_prehole* occurs we transition into *guarded_cartesian* which is a cartesian motion towards the hole with monitors that trigger when end-effector force exceeds a threshold. This procedure of reconfiguring the goals of the atomic skill is adhered to for all of the transitions.

TABLE I
PROTOTYPE DESCRIPTION

Part name	Tasks	Composed Skill	Atomic Skills	Tool
small_rotor	*insert*	*grasp_and_insert*	*move_cartesian guarded_cartesian cylinder_insert*	*gripper*
big_rotor	*insert*	*grasp_and_insert*	*move_cartesian guarded_cartesian cylinder_insert*	*gripper*
housing_lid	*place*	*grasp_and_place*	*move_cartesian guarded_cartesian*	*gripper*

As described in Section II, each atomic skill is implemented as an eTaSL script. This script contains a number of constraints, monitors and input/output ports. For example, in the *cylinder_insert* skill, a constraint is used to impose a specified insertion force. The constraint is parameterized by the instantaneous task frame and desired insertion force $F_{z,des}$ along the cylindrical axis. Meanwhile zero forces and torques are maintained along and around the other axes. The location of the instantaneous task frame is a *FeatureFrame* instance created on the peg-face of the *Insert* instance, this is an an application parameter of the composed skill. The target insertion force is a process parameter given by the inference module. The dither is applied as superposed torques around the x and y axes of the instantaneous task frame. The selection of the dither axes is based on empirical results.

D. Inference Module

A Mamdani-type fuzzy inference is chosen and implemented in MATLAB R2017a's fuzzy logic designer toolbox. 21 rules are defined, describing the mapping from the peg's dimension and the clearance to the appropriate insertion force and dither amplitude. The fuzzy rules are derived from rough modeling of the contact situation and the empirical trend observed after executing the *cylinder_insert* skill. Each fuzzy rule is approximated with gaussian membership functions

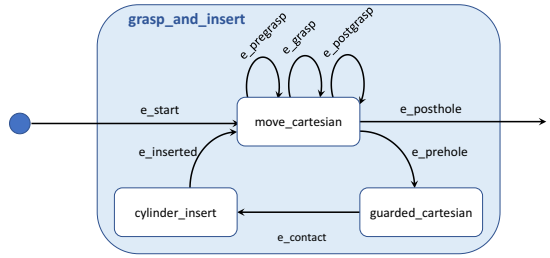

Fig. 5. Example of the finite state machine for *grasp_and_insert*. The transition between two states is triggered by incoming event "e_event_name".

that categorize a range of parameter values into fuzzy sets. For example, the peg's diameter can be categorized into "*very small*", "*small*", "*medium*", "*large*" and "*very large*". The fuzzy sets for each geometric and process parameters are shown in Fig. 6 while five sample rules are provided in Listing 1.

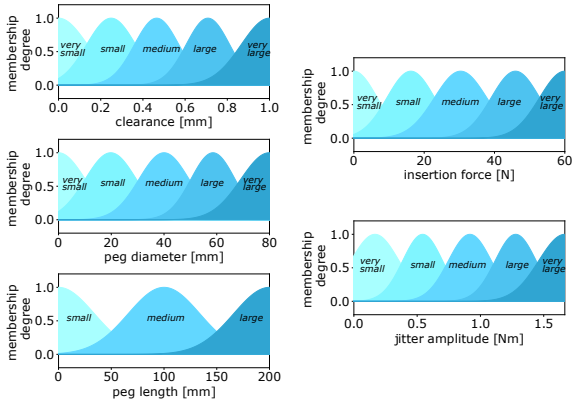

Fig. 6. The fuzzy sets used in the inference module. The left column shows the geometric parameters while the right one shows the process parameters.

Listing 1. Selection of the fuzzy rules used in the inference module

```
if diameter is "small" then insertion force is "large"
if clearance is "medium" then insertion force is "medium"
if length is "long" then insertion force is "small"
if clearance is "very small" then dither amplitude is "very large"
if diameter is "small" and length is "long" then dither amplitude is "medium"
```

In this data-driven approach, we first collected the training data by running a total of 40 peg-in-hole trials with different process parameter values. For each run, the insertion force and dither amplitude were carefully selected to ensure successful execution, while minimizing contact force. Based on these experiments, the number of membership functions and their shapes were then tuned. Finally, the fuzzy inference module was validated with a new set of peg-in-hole tasks.

E. Control Layer

The experimental setup consists of a seven DOF KUKA LBR iiwa 14 robot equipped with a pneumatic SCHUNK RH940 parallel gripper. The robot is controlled with eTaSL/eTC [11] which is deployed as an Orocos [19] component. eTaSL is used for constraint specification in Lua language while eTC is the controller implementation that satisfies the constraints in an instantaneously optimal way. The control loop is run with a frequency of 200 Hz. Additionally, a *feature_frame_publisher* node [20] was created in ROS [21] for publishing all relevant frames with TF2. Actuation of the gripper is also performed through a ROS service call.

IV. Experimental Results

The overall system is deployed for assembling the small rotor, big rotor, and housing lid consecutively. The rotor assemblies are categorized as tight-tolerance tasks, therefore the *grasp_and_insert* composed skill using force control is selected, and process parameter inference is required. The geometric property generated from the CAD information of the tasks and their inferred process parameters are reported in Table II. Meanwhile, since the housing lid assembly has a relatively large clearance, a position control skill with force monitor i.e. *guarded_cartesian* is sufficient. Therefore, process parameter inference is not required.

The experiment shows that the robot successfully assembles the parts using a parametrized skill by the application and process layers, see Fig.8 for a snapshot of the assembly sequence. The recorded insertion forces and moments during a portion of the assembly sequence are shown in Fig 7. The plot gives a recording of the transition from *guarded_cartesian* to *cylinder_insert*. To emphasize the behavior of the dithering in the skill, we do not transition from *cylinder_insert* to *move_cartesian*, but remain in the *cylinder_insert* skill.

TABLE II
Geometric and Inferred Process Parameters for the Rotor Assemblies

Parameter Category	Parameter Name	Part Name *small_rotor*	Part Name *big_rotor*
Geometric	Length (mm)	192	192
	Diameter (mm)	61	75.5
	Clearance (mm)	0.1	0.08
Process	Insertion Force (N)	-31.1	-33.5
	dither Amplitude (Nm)	1.13	1.33

V. Discussion

The force controller in eTaSL has a damping behavior [11]. This resulted in a velocity proportional to the difference between the measured and desired force. The plot in Fig. 7 shows that when the robot approached the rotor's hole between 0 s and 2.8s the forces were approximately zero. When a force magnitude of -10 N in F_z was experienced by *guarded_cartesian* a transition into *cylinder_insert* occurred. The insertion process lasted from 2.8 s to 6.7 s and the robot was moving down with a constant velocity. The effects of friction (approximately -10 N in z-direction) are visible

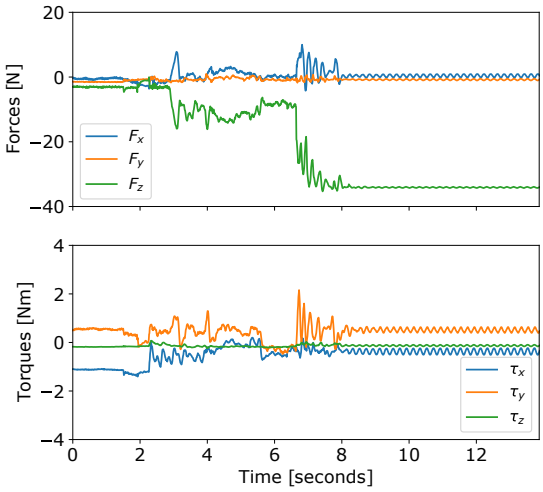

Fig. 7. The measured forces and torques during insertion of the big rotor. Note that from 8 s, the rotor is in contact with the bottom. This was added to verify that the desired dithering is present.

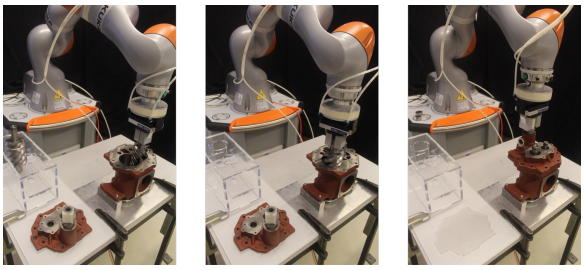

Fig. 8. The robot performing the assembly sequence of the compressor parts. From left to right: assembly of the small rotor, the big rotor, and the housing lid.

during the insertion. The remaining difference between insertion force set-point and the measured friction caused an approximately constant insertion velocity. The dithering was not easily distinguished in the graph during the insertion process. From 6.7 s to 8 s, the rotor touched the bottom of the hole, the velocity dropped to zero, and the set-point for the insertion force reached its desired value. The measured force along the z-axis was consistent with the selected process parameters listed in Tab. II. Furthermore, the oscillations due to the applied dither-force remained bounded. This shows that the skill executed the assembly task as desired.

The composed skills are given the names of feature frames. These are used to look up the location of parts using the published frames, making them less dependent on hard-coded locations. An example of this is the grasp frame available on each of the parts. The pre-grasp and post-grasp locations were constant offsets from the grasp frame, and the grasp frame was queried from TF2 based on the part name. This allows easy integration with robot cell localization systems that publish part locations using TF2.

The published frames were also beneficial in calibration of the robot cell as we could use the known CAD data on where the holes are on the parts to quickly calibrate the part location, e.g. we moved the end-effector to the grasp location of a rotor and read where the mesh was located relative to the grasp location from the *feature_frame_publisher*. This can be useful if a teach-in of the robot cell information is desired.

An aspect of the geometric information that was not incorporated was the part symmetry, the rotors are cylindrical and can be grasped or inserted from any orientation around its z-axis. This information is available in the JSON file of the part, and requires developing an atomic skill, *move_cartesian_symmetric*, where the symmetric information is added to the application parameters of the skill. This would allow more efficient execution of the approach motions. Depending on the specific mounting of the gripper on the robot, this can also increase the working range of the robot.

In this paper we have emphasized the geometric information available in the CAD data, but there is another aspect that is of relevance, the material properties of the parts involved. For example, insertion of a teflon peg in a steel hole has different process parameters to inserting a steel peg in a soft-plastic hole. Material properties are available in STEP AP214 file format, and many CAD programs support it. For multi-material assembly scenarios, the inference engine should be trained on the material properties as well as the geometric properties.

VI. Conclusion

In this paper, a three-layered system architecture for automating robot assembly programming using CAD information is proposed. The application layer is used for annotating the tasks, the process layer is used for planning and inference of the relevant robot skills, and the control layer is used for execution of the skills on the robot platform. We focus on more challenging assembly tasks that require force-control. To accomplish this we use skills that have process parameters such as insertion force and dither amplitude. These empirical parameters are not readily available in the CAD model. A fuzzy logic inference module is used to capture this process knowledge by providing a method of inferring the process parameters based on a set of key geometric parameters from the CAD model.

As a proof-of-concept, we implemented the proposed system architecture and validated it with force-controlled insertion of compressor rotors. The problem can be generalized as a peg-in-hole task, for which the conventional solutions have been studied well [22]. The key geometric parameters used in the fuzzy logic was the peg's clearance, peg's diameter, and insertion length, all of which were extracted from the CAD model and Task object in the application layer. For more complex assembly tasks, such as click-connection, the key geometric parameters are yet to be defined.

Inference is realized using a fuzzy inference method which proved to be straightforward to implement. Of course, this

method fails to extrapolate outside the predefined range of the fuzzy sets and we assumed that all of the given geometric parameters are relevant. With a sufficiently large dataset and more advanced regression methods (such as automatic relevance detection [23]), it is possible to deduce which of the parameters are really relevant for the process parameters. Such an approach will be beneficial for including material properties when inferring process parameters. The fuzzy set database can grow over time to accommodate a larger variety of assembly cases and this database can be shared with similar robot setups.

For many industrial assembly cases, there are more than two contact surfaces involved. An example is the housing lid and the two rotors. There are two main approaches to this: defining a primitive boundary representation [4], or using the subshape. In this article we have assumed the chosen geometric primitive to be sufficient for completing the task. To handle more complex geometries, a task could be split into subtasks with their individual primitive boundary representations, or a special purpose composed skill can be defined.

The preliminary implementation contains prototypes and tools that the eventual system will use. The assembly of the rotors and housing lid was successfully executed by the system, and the inference module provided reasonable parameters. The application layer implementation is useful for annotating geometric features to CAD models and the publisher node gives their transformations in ROS.

A wealth of research exists on assembly analysis, planning, and skills. This research can become tools that can benefit researchers as well as manufacturers. We describe a system architecture based on previous work that also allows for process parameter inference for tight-tolerance assembly situations and share some of the tools built in the process. However, it is difficult to build up a comprehensive system. Therefore we advocate prototyping in open-source frameworks such as FreeCAD and Orocos.

VII. Acknowledgement

The work reported in this paper was supported by Flanders Make ICON FINROP (Fast and Intuitive Robot Programming) in Belgium and the centre for research based innovation SFI Manufacturing in Norway. The work is partially funded by the Research Council of Norway under contract number 237900.

References

[1] A. Ambler and R. Popplestone, "Inferring the positions of bodies from specified spatial relationships," *Artificial Intelligence*, vol. 6, no. 2, pp. 157–174, jun 1975.

[2] L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM Journal of Research and Development*, vol. 21, no. 4, pp. 321–333, jul 1977.

[3] S. Kaufman, R. Wilson, R. Jones, T. Calton, and A. Ames, "The Archimedes 2 mechanical assembly planning system," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4. IEEE, 1996, pp. 3361–3368.

[4] U. Thomas and F. M. Wahl, "Assembly Planning and Task Planning — Two Prerequisites for Automated Robot Programming," in *Springer Tracts in Advanced Robotics*, 2010, vol. 67, pp. 333–354.

[5] F. Dietrich, J. Maaß, A. Raatz, and J. Hesselbach, "RCA562: Control Architecture for Parallel Kinematic Robots," in *Springer Tracts in Advanced Robotics*, 2010, vol. 67, pp. 315–331.

[6] P. Neto, N. Mendes, R. Araújo, J. Norberto Pires, and A. Paulo Moreira, "High-level robot programming based on CAD: dealing with unpredictable environments," *Industrial Robot: An International Journal*, vol. 39, no. 3, pp. 294–303, apr 2012.

[7] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2016-Novem. IEEE, oct 2016, pp. 2293–2300.

[8] M. T. Mason, "Compliance and Force Control for Computer Controlled Manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 6, pp. 418–432, 1981.

[9] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, 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. 26, no. 5, pp. 433–455, may 2007.

[10] 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.

[11] E. Aertbelien and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, sep 2014, pp. 1540–1546.

[12] M. Klotzbücher and H. Bruyninckx, "Coordinating robotic tasks and systems with rfsm statecharts," *JOSER: Journal of Software Engineering for Robotics*, vol. 3, pp. 28–56, 2010.

[13] T. De Fazio and D. Whitney, "Simplified generation of all mechanical assembly sequences," *IEEE Journal on Robotics and Automation*, vol. 3, no. 6, pp. 640–658, dec 1987.

[14] L. Homem de Mello and A. Sanderson, "Representations of mechanical assembly sequences," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 211–227, apr 1991.

[15] N. Pitchandi, S. P. Subramanian, and M. Irulappan, "Insertion force analysis of compliantly supported peg-in-hole assembly," *Assembly Automation*, vol. 37, no. 3, pp. 285–295, 2017. [Online]. Available: http://www.emeraldinsight.com/doi/10.1108/AA-12-2016-167

[16] S. Guillaume, "Designing fuzzy inference systems from data: An interpretability-oriented review," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 3, pp. 426–443, jun 2001.

[17] J. Riegel and Y. van Havre, "FreeCAD: Parametric 3D modeler." [Online]. Available: https://www.freecadweb.org

[18] M. H. Arbo and Y. Pane, "ARBench," 2017. [Online]. Available: https://github.com/mahaarbo/ARBench

[19] P. Soetens, "A Software Framework for Real-Time and Distributed Robot and Machine Control," Ph.D. dissertation, Katholieke Universiteit Leuven, 2006.

[20] M. H. Arbo and Y. Pane, "ARBench part publisher," 2017. [Online]. Available: https://github.com/mahaarbo/arbench{_}part{_}publisher

[21] M. Morgan Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, may 2009.

[22] T. Lozano-Perez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *The International Journal of Robotics Research*, vol. 3, no. 1, pp. 3–24, 1984.

[23] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.