VTT Technical Research Centre of Finland

Programming and control for skill-based robots

Saukkoriipi, Janne; Heikkilä, Tapio; Ahola, Jari M.; Seppälä, Tuomas; Isto, Pekka

Published in:
Open Engineering

DOI:
10.1515/eng-2020-0037

Published: 01/01/2020

Document Version
Publisher's final version

Research Article

Janne Saukkoriipi*, Tapio Heikkilä, Jari M. Ahola, Tuomas Seppälä, and Pekka Isto

Programming and control for skill-based robots

https://doi.org/10.1515/eng-2020-0037
Received Jul 02, 2019; accepted Feb 27, 2020

Abstract: This paper considers programming and control of skill-based robots. Robot skills are used to integrate and synchronize robot actions and sensor data in a consistent way. Skill-based approach provides a framework for configurable robot systems, enabling quick setups and start-ups of applications. In the paper we will introduce skill programming and skill control concepts in more detail and how they relate to usage of models and sensors. We will also give a practical example for programming and implementing skills for a grinding application.

Keywords: robot skill, programming, sensor, control

1 Introduction

With ever-increasing demands for agility and flexibility, robotized manufacturing is facing challenges and barriers to the use of robots because; the high cost of engineering the work cells (i.e., the design, fabrication, and installation of jigs, fixtures, conveyors, and third-party sensors and software). Robots must be able to perform their tasks in environments with greater uncertainty than current systems can tolerate [1], where the robustness implies wide usage of a variety of sensor technologies. Robust robot operations can be based on behavioral models supporting the integration of sensors into robot operations - this is how the robot "skills" are formulated. A skill is typically defined as a high level entity as a sensory-motoric decomposition pattern, where skills are decomposed to action and trajectory level representations [2]. Skill representations support reusability and when the skill layer is abstract it provides a non-expert and intuitive programming interface, but still relies on a device level setup by an expert user [3]. Reusability is two-fold, it can be supported by simpler, or conversely by more complex skills [4]:

* simpler skills are easier to re-use and may be related to generic description on a symbol level, but
* hardware independence can only be achieved to a certain degree, but is more difficult if skills are simple.

Skills can be hierarchic and at the lowest level there are atomic actions representing device interfaces or API functions. Hierarchic structure in skill-based control and programming will go as follows: a "task" is described as the highest-level of operation like for example "Single part feeding", "Welding with localization" and "Pick-up with contact-force" [5]. A task can be decomposed to "skills" such as "Locate box", "Pick-up box" and "Place box" [5–7], which are formed from the lowest-level entities called "primitives", which can be described as "device operations". Further, the "primitive" layer can be seen to represent as the conventional robot work sequence, consisting of commands such as "Move linear", "Open gripper", "Get point-cloud" and "Get force-values". Skills should be designed in a way, where parametrization supports an easy and quick re-configurability of the skills for different situations in varying applications, for example phase dependent locations/points, poses, forces and so on.

To our knowledge, there are no off-line programming tools for skill based robot systems, where both robot path programming - especially with compliant motion control primitives - and sensor programming were integrated. Schou *et al.* [8] presents a similar hierarchical functional modelling and parametrization approach to task-level programming. But in their system, only some parameters were specified off-line (e.g. velocity, object type etc.) and all the locations were taught on-line. Whereas in our system, all skills are fully programmed off-line (with only updating the specific parameters based on information gained from the other skills. In their paper, Schou *et al.* also differentiate computational procedures from skills to "services", which is a challenging generalization since almost every task, skill and primitive include some sort of computational procedures which can be implemented as SW services.

In addition, as we have been developing robot skills especially relying on vision feedback and compliant contact motions, and for that there are only few, if any, reported

***Corresponding Author: Janne Saukkoriipi:** VTT Technical Research Centre of Finland Ltd; Email: janne.saukkoriipi@vtt.fi
Tapio Heikkilä, Jari M. Ahola, Tuomas Seppälä, Pekka Isto: VTT Technical Research Centre of Finland Ltd, Tapio Heikkilä, Jari M. Ahola, Tuomas Seppälä, and Pekka Isto

references in the literature. Zeiss has reported on modeling principles [9], where force and torque have been interconnected in the task representation with the a pose-wrench approach (wrench=force /torque parameters) but it considers only static path points and not compliance control.

In this paper we will introduce methods and tools for robot skill programming and an example of skill based control system. Skill execution control - or dynamics - is specified as UML action diagrams, the implementation of which may be distributed and implemented as robot programs or PLC programs. Skills are specified with (configuration) parameters, and our integrated tool supports setting and computing these, using a robot off-line programming tool with our extensions. In the experiments we will show examples of skills that rely heavily on off-line programming methods.

2 Skill models and implementations

Ideally, skill models are designed and programmed by expert users but they could be re-configured for different situations even by operators who have no experience in robot programming. This is achieved by making skills intuitive object-centered robot abilities, where they are applied on physical entities instead of 3D coordinates [5]. In practice, this can be accomplished with a proper parametrization of programs, where a set of inputs will define the whole structure of a program. The program would automatically give all the necessary parameters for the sensors and control systems in a system. Robot skills are divided to primitive skills and compound skills, composed of primitive skills and other compound skills. These are described further below.

2.1 Primitive skills

Feature based localization

Fundamentally, rough global localization of the target object is needed to produce the initial pose parameters for the fine localization. Rough location of the object can be determined based on known features of the object e.g. eigenvectors and center points of the 3D point clusters, or other known vectors, like surface normals or interest points, like corners. The non-orthogonal estimate of the rotation matrix Re can be computed as

$$Re = N_2 N_1^+ \tag{1}$$

in which N_1 and N_2 are matrices including surface normal vectors and have more columns than rows. N_1^+ denotes pseudoinverse which can be computed as

$$N_1^+ = N_1^T \left(N_1 N_1^T\right)^{-1} \tag{2}$$

required that matrix $N_1 N_1^T$ is invertible. The matrix N_1 include vectors in the object's CAD model coordinate system

$$N_1 = \begin{bmatrix} \overline{n}_1 & \dots & \overline{n}_i \end{bmatrix} \tag{3}$$

and the matrix N_2 include vectors in the sensor's coordinate system

$$N_2 = \begin{bmatrix} \overline{n}_{m1} & \dots & \overline{n}_{mi} \end{bmatrix} \tag{4}$$

The orthogonal estimate of the rotation matrix is constructed from Re by taking cross products of the columns of the rotation matrix estimate

$$\overline{v}_z = \overline{v}_x \times \overline{v}_y \tag{5}$$

in which $\overline{v}_x = Re_{1:3,1}$ and $\overline{v}_y = Re_{1:3,2}$.

$$\overline{v}_x = \overline{v}_y \times \overline{v}_z \tag{6}$$

The initial, i.e. rough, rotation matrix is constructed from the orthogonal unit vectors.

$$R_{init} = \begin{bmatrix} \frac{\overline{v}_x}{|\overline{v}_x|} & \frac{\overline{v}_y}{|\overline{v}_y|} & \frac{\overline{v}_z}{|\overline{v}_z|} \end{bmatrix} \tag{7}$$

The initial translational vector T_{init} is computed as

$$\overline{T}_{init} = \overline{p}_{c2} - R_{init}\overline{p}_{c1} \tag{8}$$

in which p_{c1} is the corresponding center point in the object's CAD coordinate system and p_{c2} is the cluster center point in the sensor's coordinate system.

Verification vision

In the fine localization, the target object's pose is estimated by fitting surface models to the measured 3D-points. Here the equations for fitting planar surfaces are presented and equations for fitting other surface types have been presented in [14]. The rotation correction ΔR is computed as a product of rotation matrices

$$\Delta\mathbf{R} = \mathrm{Rot(x, \Delta\phi_x)Rot(y, \Delta\phi_y)Rot(z, \Delta\phi_z)} \tag{9}$$

in which

$$Rot(x, \Delta\phi_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\Delta\phi_x) & -\sin(\Delta\phi_x) \\ 0 & \sin(\Delta\phi_x) & \cos(\Delta\phi_x) \end{bmatrix} \tag{10}$$

$$Rot(y, \Delta\phi_y) = \begin{bmatrix} \cos(\Delta\phi_y) & 0 & \sin(\Delta\phi_y) \\ 0 & 1 & 0 \\ -\sin(\Delta\phi_y) & 0 & \cos(\Delta\phi_y) \end{bmatrix} \quad (11)$$

$$Rot(z, \Delta\phi_y) = \begin{bmatrix} \cos(\Delta\phi_z) & -\sin(\Delta\phi_z) & 0 \\ \sin(\Delta\phi_z) & \cos(\Delta\phi_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

where $\Delta\varphi_x$, $\Delta\varphi_y$, and $\Delta\varphi_z$ are the rotation corrections with respect to the coordinate axes.

The translational correction ΔT is defined by three translational components

$$\Delta\mathbf{T} = \begin{bmatrix} \Delta t_x \\ \Delta t_y \\ \Delta t_z \end{bmatrix} \quad (13)$$

The correction vector Δ is defined as

$$\Delta = \begin{bmatrix} \Delta\phi_x \\ \Delta\phi_y \\ \Delta\phi_z \\ \Delta t_x \\ \Delta t_y \\ \Delta t_z \end{bmatrix} \quad (14)$$

The correction vector is calculated

$$\Delta = -(J^T J)^{-1} J^T \cdot E \quad (15)$$

in which J is the Jacobian matrix gathered from partial derivatives of the error functions e_i with respect to correction vector Δ.

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial \Delta} \\ \vdots \\ \frac{\partial e_N}{\partial \Delta} \end{bmatrix} \quad (16)$$

E is the error vector.

$$E = \begin{bmatrix} e_1 \\ \vdots \\ e_N \end{bmatrix} \quad (17)$$

The partial derivatives are computed as follows

$$\frac{\partial e_i}{\partial \Delta} = \frac{\partial e_i}{\partial p_{Mi}} \frac{\partial p_{Mi}}{\partial \Delta} \quad (18)$$

If the reference feature is a plane the error e_i is computed as

$$e_i = \frac{(\mathbf{p}_{Mi} - \mathbf{p}_p)^T \mathbf{n}_p}{\sqrt{\mathbf{n}_p^T \mathbf{n}_p}} \quad (19)$$

in which p_{Mi} is the i:th measured point and the plane surface is defined by point p_p and normal n_p. The partial derivative of e_i respect to p_{Mi} is

$$\frac{\partial e_i}{\partial \boldsymbol{p}_{Mi}} = \frac{1}{\sqrt{\boldsymbol{n}_T^T \boldsymbol{n}_T}} \boldsymbol{n}_T^T \quad (20)$$

The partial derivative of p_{Mi} with respect to Δ is

$$\frac{\partial \mathbf{p}_{Mi}}{\partial \mathbf{\Delta}} = \begin{bmatrix} \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta\phi_x} & \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta\phi_y} & \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta\phi_z} & \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta t_x} & \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta t_y} & \frac{\partial \mathbf{p}_{Mi}}{\partial \Delta t_z} \end{bmatrix} \quad (21)$$

$$\frac{\partial p_{Mi}}{\partial \Delta} = \begin{bmatrix} 0 & -z_{Mi} & y_{Mi} & -1 & 0 & 0 \\ z_{Mi} & 0 & -x_{Mi} & 0 & -1 & 0 \\ -y_{Mi} & x_{Mi} & 0 & 0 & 0 & -1 \end{bmatrix} \quad (22)$$

The updated T and R of the target object pose are calculated as

$$\mathrm{T} = \mathrm{R}\Delta\mathrm{T} + \mathrm{T} \quad (23)$$

$$\mathrm{R} = \mathrm{R}\Delta\mathrm{R} \quad (24)$$

Compliant motion control

The compliant motion control primitive, "Cartesian impedance motion control" lays the basis for motion accuracy and the compliant contact motions. The impedance controller regulates the relation between robot's position x and contact force F according to time domain equation

$$F = M_t(\ddot{x} - \ddot{x}_0) + B_t(\dot{x} - \dot{x}_0) + K_t(x - x_0) \quad (25)$$

in which M_t is the target mass, B_t is the target damping, K_t is the target stiffness, x_0 is the nominal position trajectory and x is the actual position. A schematic diagram for the corresponding one degree-of-freedom impedance controlled motion is given in Figure 3.

The feasible values for target model parameters depend on the dynamics of the robot, stiffness of the environment and the characteristics of the contact task. Examples of derived impedance parameters for two target systems are [14]:

- Parameters for hard contact (K_e = 100*10^3 N/m)
  - Target model frequency ω_t = 0.8 * 2 Hz = 1.6 Hz
  - Target stiffness K_t = 100 N / 0.01 m =10000 N/m.
  - Target mass M_t = 10000 N/m / (3.2π rad/s)^2 = 98.95 kg
  - Target damping ξ_t = 14.75 and B_t = 29351 Ns/m.
  - Target impedance model $G_t = 98.95s^2 + 29351s + 10000$
- Parameters for soft contact (K_e = 1000 N/m)
  - Target stiffness K_t = 0 N/m (i.e. mass-damper)
  - Target mass M_t = 20 N / 1 m/s^2 = 20 kg
  - Target damping ξ_t = 1.0 and B_t = 282.0 Ns/m (critically damped)

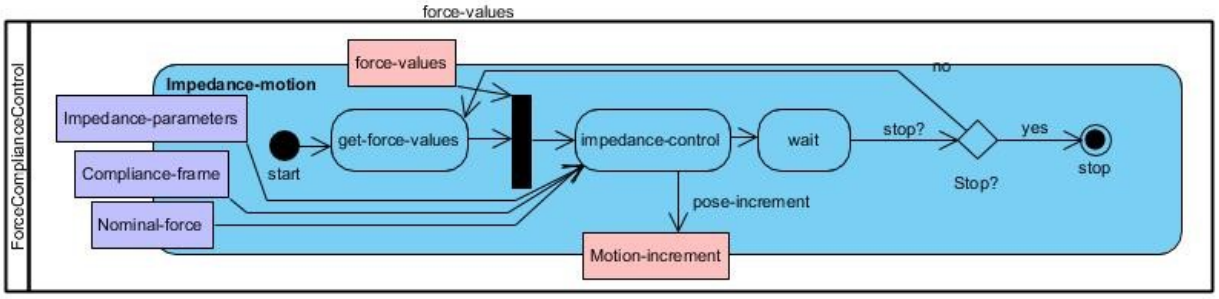

Figure 1: Primitive skill: impedance controlled contact motion

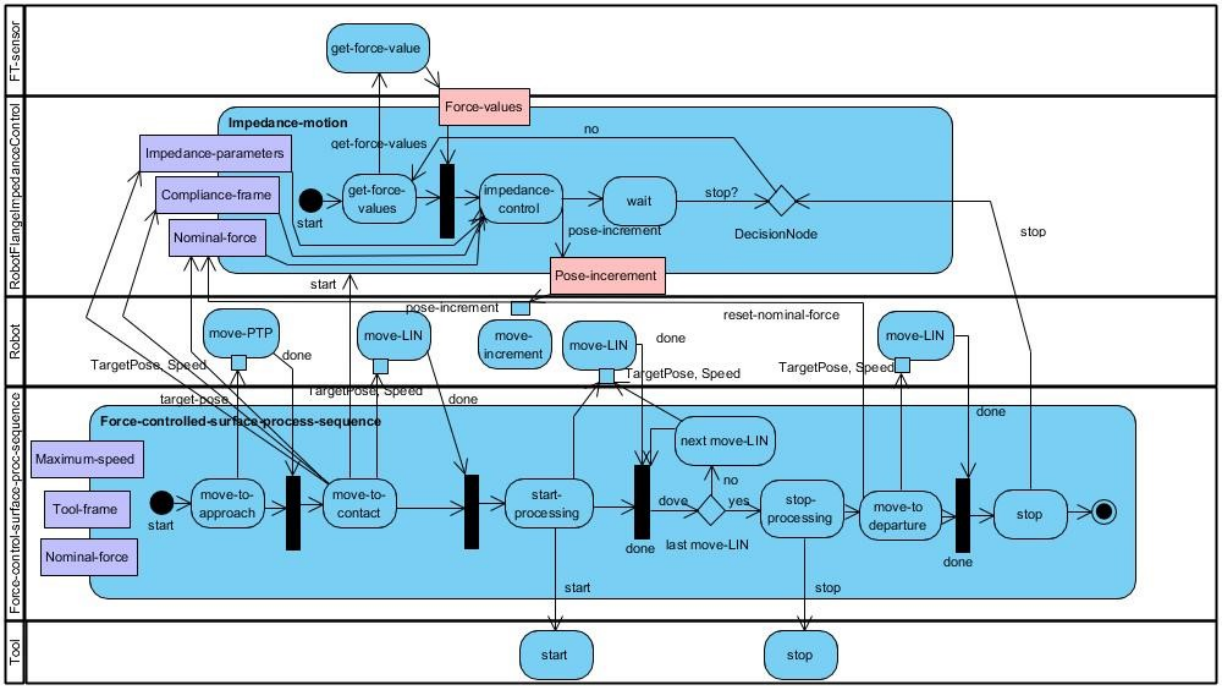

Figure 2: Compound skill: grinding of an object feature, like plane or edge, with contact to followed object surfaces

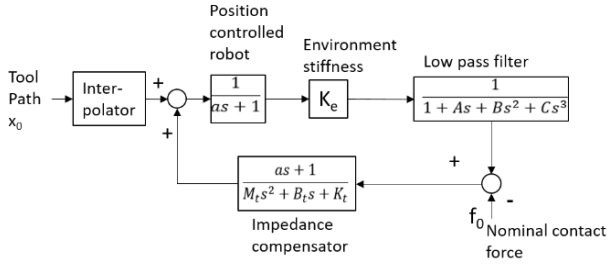

Figure 3: The schematic control system transfer functions of the impedance controlled robot

– Target impedance model $G_t = 20s^2 + 282s$

A detailed description of designing robust impedance controller for the industrial robot arm is given in [14] and an analysis of the impedance controller's capability to follow inclined and curved surfaces while keeping the contact force in desired limits is given in [15].

2.2 Skill modeling and compound skills

The compliant motion control primitive, "Cartesian impedance motion control" is outlined as a UML activity model in Figure 1. It has two types of input parameters. The setup parameters, the "configuration" parameters are nominal force, compliance frame, and impedance control parameters (stiffness, damping and inertia). In addition, it has control parameters, i.e. measured forces/torques and as output parameters, the pose increments. As a sequence, it is simple: monitor the contact forces and torques, execute the impedance control compensator, and execute the incremental Cartesian motion increments with the robot.

Typically, the compliance frame coincides the tool frame but certain tasks may require offsetting the compliance frame respect to the tool frame. The nominal force is a force offset that deviates the position of the robot to the desired direction in compliance with the target impedance model.

The surface processing skill represents a compound skill. There are many tasks where the management and control of the tool contact to the target object is critical. The surface processing skill relies on the impedance controlled motion sub-skill. It has as setup input parameters process and tool specific nominal force, and tool coordinate frame, which further forwarded to the impedance motion control primitive (the compliance frame is set to the contact point, i.e. the tool end tip) and also the tool dependent maximum speed. The surface processing skill includes motions for approaching the target, getting into contact, executing the process motions in contact and impedance controlled, and finally departing form the target. This is outlined in the Figure 2.

2.3 Skill programming

Programming of a skill can be based on off-line or on-line programming methods. In on-line programming, skill parameters are taught to the system using the actual object in a real world. On the other hand, off-line programming methods rely on using off-line programming tools together with 3D CAD-models of robots, tools, devices, and the target objects to configure the skill parameters. Skill parameters vary with different actions; they should define the actions for all the components in a robot system. Our proposal for the data flow for skill programming is given in Figure 4. Starting from the skill model and object models (or the real object), the skill parameters are acquired, and transformed into forms needed throughout the skill hierarchy.

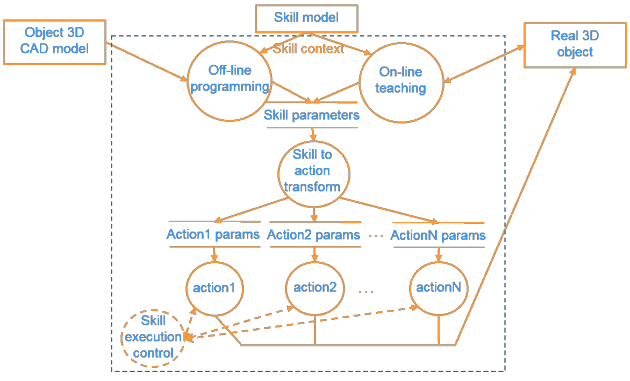

Figure 4: Skill programming can be done off-line based on geometric models or on-line based on sensor data

2.4 Skill control

The robot system control model represents the architecture, which enables the robot to integrate and synchronize sensor data and robot actions. The control architecture (Figure 5) is based on proprietary and open interfaces, the latter of which are in our case based on ROS (Robot Operating System) [10]. Proprietary interfaces are implemented for connecting devices, i.e. sensors and robot and data interfaces with adapters to ROS. Key functional components representing the objects (or "swim lanes") in the skill models, include an Object Detector component and a variety of 3D Point Cloud acquisition components (or Point Cloud sources), based on 3D cameras (based on time-of-flight or triangulation) or cooperation of a set of motion provider components (robots, transfer axis and conveyors) together with 2D laser profiler sensor components. Crosswise usage of these are supported over ROS based interfaces for 3D Point Cloud sources and 2D profiles.

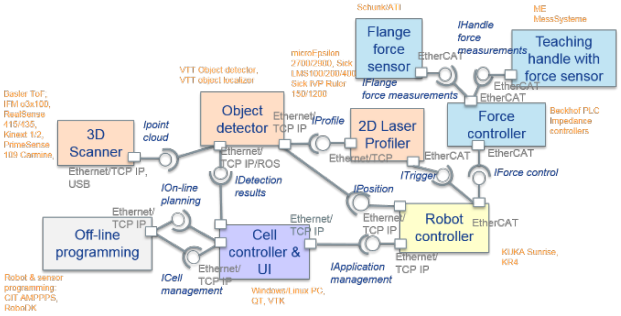

Figure 5: A flexible system architecture for integrating sensor data and robot control

3 Case example: Programming and control grinding skills

"Grinding with localization" – task is used as a case example for off-line programmed skills. It consists of lower-level skills such as global rough localization with a 3D-depth sensor, precise localization with a 2D-profile scanner and straight seam grinding. All the skills are programmed in an off-line programming tool RoboDK [11] and implemented in a robot cell, composed of KUKA KR120 R2500 PRO in-

dustrial robot with grinding tools, Intel's RealSense D415 depth sensor and microEpsilon's 2D-profile scanner. The experimental work object consisted mainly of plane-like surfaces, so all the skills were mainly designed for objects with such features. The process sequence will go as following:

1. Programming global localization
2. Programming of the precise (local) localization and related scanning motions
3. Global localization
4. Precise localization
5. Straight seam grinding

First, the work object will be localized globally in the robots workspace with accuracy of a few centimeters. For this, the sensor system needs a set of reference features, in this case a set of 3 planar surface segments and some metric parameters of these (e.g., size and nominal directions and center points). These reference features are programmed with the robot programming tool. The estimated initial position of the work object is used to program scanning movements for the robot to localize the object more accurately. This means, that the robot will use a more accurate 2D-profile scanner to scan local features of the object to get enough data for more precise localization. In precise localization phase, the measured data will be combined with reference data to produce a sub-millimeter-level accuracy for the object localization before applying the grinding motions. Finally, after the object has been localized, the grinding movements can be executed.

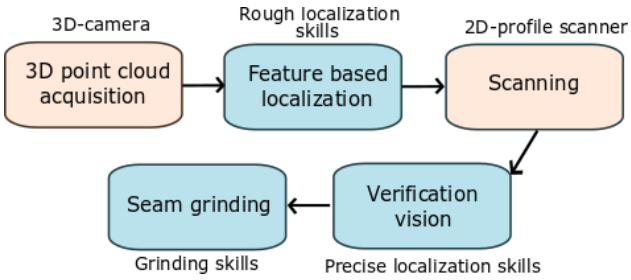

Figure 6: Skills in "grinding with localization" – task

4 Experimental results and discussions

4.1 Global localization

The aim of global localization is to locate roughly the work object within the robot workspace. Accuracy of the localization depends on the sensor used in the process. Generally, it should be within a few centimeters, depending on the view angle of the scanning sensor and scanning distance. The global localization sensor used in the experiments is Intel's RealSense D415, which is a 3D-depth sensor that uses stereo vision with infrared projector to create a depth image of the scene (see Figure 7). 3D-depth sensors are used because they offer a suitable range with reasonable accuracy for this purpose. With multiple external sensors, the whole workspace of industrial 6-DOF robot can be monitored reliably. If the object is known to be located at certain side of the robot, a single 3D-depth sensor was determined to be sufficient.

Figure 7: Colored 3D depth image captured with RealSenseD415 [12]

Localization itself was performed by comparing and matching the reference and measured surfaces. Based on the camera data, the measured surfaces can be represented in robot coordinate system and matched against the reference surfaces. On the other hand, the reference surface normals are represented in 3D-models coordinate system. Thus, the comparison of surface normals will result a position of a 3D-models coordinate system relative to the robot's coordinate system.

The RoboDK tool was used to gather the reference data from 3D CAD-model; it was already used to program scanning and grinding movements for the robot. The surface data (a 3D point and a surface normal) were obtained by selecting sets of three points, one set from each of three reference surfaces of the 3D-model and calculating the nor-

mal and reference points for every surface (see Figure 8). Surface normals and related 3D points from three non-parallel surfaces needed to be gathered to express unambiguously position and orientation of the object. Adding the points and calculation of the normal vectors and the 3D points were done through Python API. On the other hand, corresponding measured surfaces (3D points, surface normal) were solved by segmenting the measured point cloud and using principal component analysis (PCA) for the segments as seen in Figure 6. Direction of the least variance was defined as the surface normal for a segment (blue axis). Main challenge of the segmentation phase is related on finding suitable segmentation parameters, which is very dependent on the geometric surfaces of the work object.

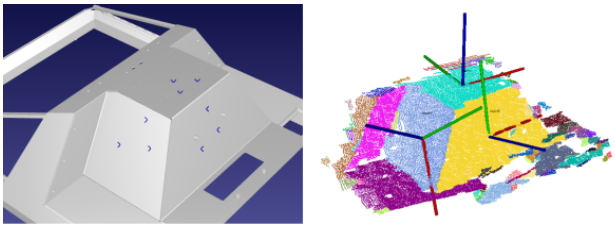

Figure 8: Gathering reference (on the left) and corresponding measured surface points and normals [12]

4.2 Precise localization

After global localization had given an estimate of the object's location and orientation, scanning movements were updated based on programmed nominal paths. Scanning movements were composed of using a robot with a 2D-profile scanner attached to its flange, performing scanning motions in proximity of the object. 2D-profile scanners were used due to their high-level accuracy, which increases accuracy of the localization significantly. Scanning movements resulted a 3D-point cloud of the features that were within the sensors measurement range, which was then filtered and segmented into different regions using Point Cloud Library's [13] point cloud processing algorithms.

After filtering and segmentation, different regions of the point cloud represented different planes of the object. The same reference surfaces (obtained the same way as for global localization) were sent to the pose estimator that was integrated into same software, which was used to capture and process the point cloud. Pose estimator corrects iteratively the accurate position parameters so that the distance between the measured point and corresponding surface in 3D- model will be minimized. The pose estimator provides a more accurate pose of the target object in the robot coordinate system, which was used to update position of the target object in the robot program ("user coordinates"). Scanning movements, processed point cloud and fitted point cloud can be seen in Figure 9 and Figure 10.

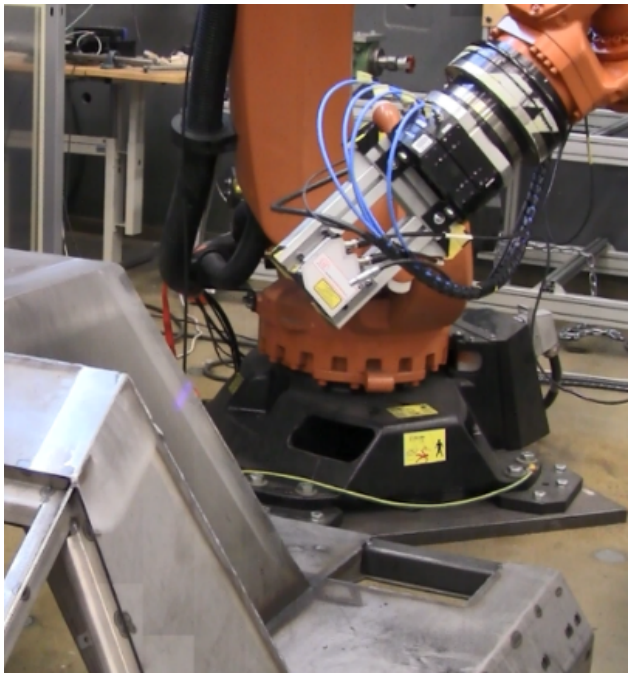

Figure 9: Scanning the object with microEpsilon 2900-100/BL profile scanner [12]

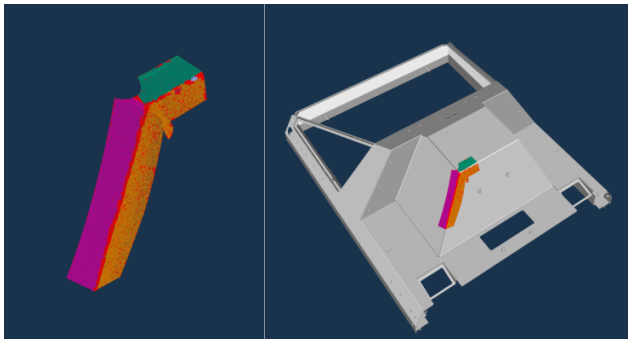

Figure 10: Segmented point cloud in precise localization, obtained by scanning motions and fitted into 3D-model [12]

4.3 Grinding skills

Even though the object was quite accurately localized after precise localization, some localization errors are still present. Even with high-level equipment and calibration routines, errors will occur due to manufacturing defects that will make the actual object to deviate from what is derived based on the 3D CAD-model. These localization errors were compensated by applying impedance-controlled compliant motions for the grinding movements. In these force-controlled motions, the robot used a force/torque sensor integrated into its flange to measure the contact forces and torques between the object and the tool. Besides encountering localization errors, it was used to enhance even the effect of grinding by applying nominal force to the surface.

Seam grinding skill is based on four input parameters, i.e. target frames that define the characteristics of the seam. Targets were placed to the end regions of the seam, as shown in Figure 11, where target frame pairs were in the start and end of the seam on adjacent surfaces. All the approach and grinding movements were based on these four input target frames. Target frames were placed easily in RoboDK by using a "Teach target(s) on the surface" feature, which sets a target frame so that the Z-axis is perpendicular to the surface and pointing downwards. After this, targets were only needed to be fine-tuned in location and orientation around Z-axis. After the targets had been placed, the paths of the grinding skill were then created and simulated (see Figure 12). The simulation showed if the path could not be completed due to crossing singularities or issues with reachability. All of this made the skill configuration and validating the program for another seam relatively easy and fast.

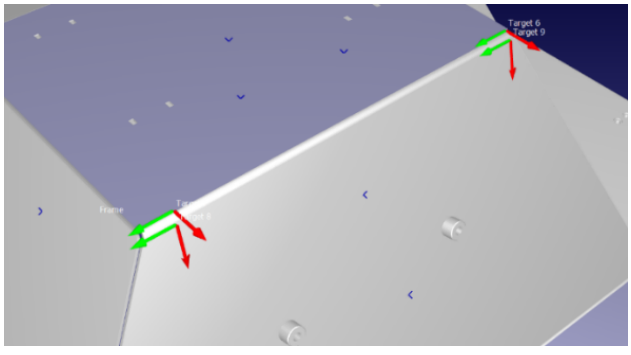

Figure 11: Defining location and orientation of the seam [12]

Seam grinding was designed to be in two phases. In the first phase, the seam will be grinded with long vertical movements with angle grinder to remove all excessive material from the seam. Objective of this phase is to result a sharp-pointed seam without any extra material left from welding. In the second phase, the seam will then be rounded by using a sander. Both phases will use the same four input targets to form the grinding programs. Results can be seen in Figure 13.

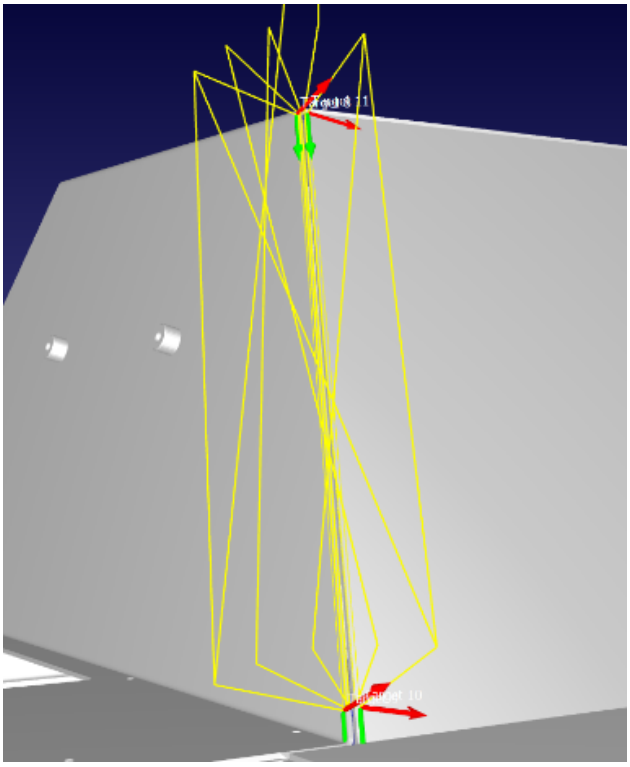

Figure 12: Full path of the grinding program visualized [12]

Figure 13: Grinding skill in different phases in left to right: seam after welding, seam after vertical grinding and fully grinded seam [12]

5 Summary

In this paper we have shown the design and implementation of off-line programmed skills. Skills such as global localization, scanning movements, precise localization and straight seam grinding were used to localize and grind the object. Programmed skills were tested successfully in a real robot environment.

References

[1] A Roadmap for U.S. Robotics (2013) - From Internet to Robotics, 2013 Edition, 129 p. Available in: http://archive2.cra.org/ccc/files/docs/2013-Robotics-Roadmap

[2] Boada MJ, Barber R, Salichs MA. Visual approach skill for a mobile robot using learning and fusion of simple skills. Robot Auton Syst. 2002;38(3-4):157–70.

[3] Andersen RH, Solund T, Hallam J. Definition and Initial Case-Based Evaluation of Hardware-Robot Skills for Industrial Robotic Co-Workers, Conf. ISR ROBOTIK; 2014.

[4] Weser M, Zhang J. Autonomous Planning for Mobile Manipulation Services Based on Multi-Level Robot Skills, The 2009 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Oct 11-15, 2009, St. Louis, USA.

[5] Pedersen MR, Nalpantidis L, Andersen RS, Schou C, Bøgh S, Krüger V, et al. Robot skills for manufacturing: from concept to industrial deployment. Robot Comput-Integr Manuf. 2016;37:282–91.

[6] Heikkilä T, Ahola JM. Robot skills – modeling and control aspects. *14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*. July 2-4 2018, Oulu, Finland

[7] Morrow J, Khosla P. Manipulation Task Primitives for Composing Robot Skills. Proc of the 1997 IEEE Int'l Conf on Robotics and Automation, Albuquerque, New Mexico. https://doi.org/10.1109/ROBOT.1997.606800.

[8] Schou C, Andersen RS, Chrysostomou D, Bøgh S, Madsen O. Skill-based instruction of collaborative robots in industrial settings. Robot Comput-Integr Manuf. 2018;53:72–80.

[9] Stphan Z. Manipulation Skill for Robotic Assembly. MsC Thesis, Technical University of Darmstadt, 2014. 117p.

[10] About ROS [Internet]. [cited 2019 Feb 7] Available from: http://www.ros.org/.

[11] RoboDK. [Internet]. [2019; cited 2019 Feb 7] Available from: https://robodk.com/

[12] Saukkoriipi J. Design and implementation of robot skill programming and control, Master's Thesis 2019, University of Oulu, Finland.

[13] Point cloud library. [Internet] [2018; cited 2019 Feb 4] Available from: http://pointclouds.org/.

[14] Heikkilä T, Ahola JM, Viljamaa E, Järviluoma M. An interactive 3D sensor system and its programming for target localising in robotics applications. Proceedings of the IASTED international Conference Robotics 2010, Robo 2010. IASTED, p. 89-96.

[15] Ahola JM, Koskinen J, Seppälä T, Heikkilä T. Development of impedance control for human/robot interactive handling of heavy parts and loads. 1 Jan 2015. 2015 ASME International Conference on Mechatronics and Embedded Systems and Applications. American Society of Mechanical Engineers ASME. Vol 9.