Skill-based Robot Programming in Mixed Reality with Ad-hoc Validation Using a Force-enabled Digital Twin

Jan Krieglstein, Gesche Held, Balázs A. Bálint, Frank Nägele and Werner Kraus

Abstract— **Skill-based programming has proven to be advantageous for assembly tasks, but still requires expert knowledge, especially for force-controlled applications. However, it is error-prone due to the multitude of parameters, e.g. different coordinate frames and either position-, velocity- or force-controlled motions on the axes of a frame. We propose a mixed reality based solution, which systematically visualizes the geometric constraints of advanced high-level skills directly in the real-world robotic environment and provides a user interface to create applications efficiently and safely in mixed reality. Therefore, state-machine information is also visualized, and a holographic digital twin allows the user to ad-hoc validate the program via force-enabled simulation. The approach is evaluated on a top hat rail mounting task, proving the capability of the system to handle advanced assembly programming tasks efficiently and tangibly.**

I. INTRODUCTION

While the number of robots used in industry is growing each year by about 10-20%, the portion of robots performing assembly tasks is staying relatively low at around 12%, and the amount of automation in assembly is still at a low level [1]. One main reason is that assembly tasks are often difficult to automate since they frequently require some sensitiveness, e.g. to overcome component tolerances or to perform force-sensitive tasks like clipping or placing. Also, the diversity of variants hinders companies from automating assembly lines. Another reason is that along with Industry 4.0 and specialization of industries as well as products, efficient programming becomes crucial, especially for small and medium enterprises (SMEs). Classical robot waypoint programming cannot address these needs.

One approach to overcome these issues is skill-based programming, providing a library of well-defined function blocks, so-called skills, which need to be parameterized. This still requires expert knowledge, and handling different coordinate frames, axes, and waypoints remains complex and error-prone, even for experienced programmers, since the programmer needs to handle motions in up to 6 degrees of freedom (DOF).

Mixed Reality (MR) can address this issues by enabling the user to act within a three-dimensional environment and to interact with the environment itself (Fig. 1). It keeps the capabilities of Virtual Reality (VR), which is to have a 3D visualization of the robot, without the need to model the whole scene. This allows to stay closer to the real-world application, drastically lowering or even avoiding the sim2real gap.

This work was supported by the Ministry of Economic Affairs of the state Baden-Württemberg in Germany under AI Innovation Center (Learning Systems and Cognitive Robotics) – Grant No. 017-180036.

All authors are with the Fraunhofer Institute for Manufacturing Engineering and Automation IPA, 70569 Stuttgart, Germany {jsk, jsk-gh, bab, frn, wek}@ipa.fraunhofer.de

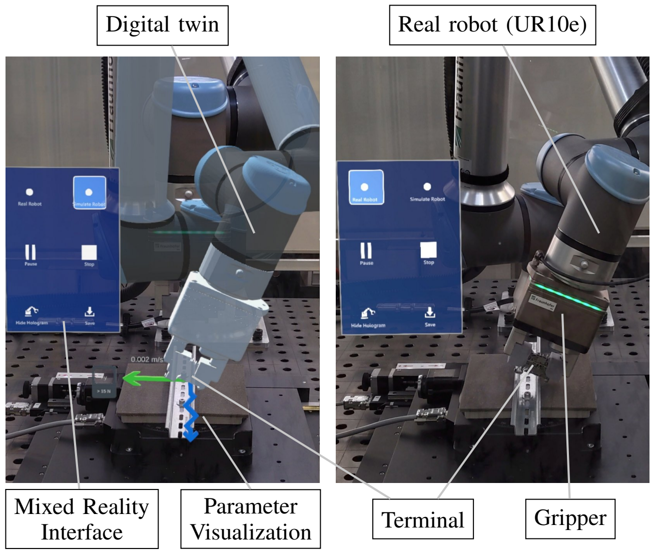


Fig. 1: Skill parameter visualization in mixed reality for a top hat rail mounting task. The interface allows on-the-fly execution on the digital twin, before it can be executed on the real robot.

This work proposes a way how MR can systematically and beneficially be used for skill-based programming: by visualizing the parameters of the skills, providing an MR-user interface for programming and an execution panel, and enabling ad-hoc validation using a digital twin with contact force simulation.

This paper is structured as follows: Sec. II summarizes the state of the art of skill-based programming and robot programming in augmented or mixed reality. Sec. III gives an overview of the skill model and selected complex assembly skills. Sec. IV then evaluates how the skill parameters can be visualized and set using MR. Sec. V details the system architecture and implementation, while Sec. VI validates the system on an example application. Sec. VII gives a conclusion and outlook.

II. STATE OF THE ART

A. Skill-based Programming

The VDI guidelines #2860 published in 1990 provided the first guidelines for a unique and comprehensible description of assembly processes [2]. In 1992, Hasegawa et al. published the first formal descriptions of skill-based robot programs, which allowed them to address typical assembly tasks with a sequence of hybrid position-, velocity-, or force-controlled manipulation skills [3].

While in the following, one branch of research focused on the automatic creation of programs from CAD data, another branch tried to make the manual composing of robot programs more efficient by supporting the programmer. One possibility is the usage of domain-specific languages (DSL), which allow to describe the task in an appropriate way (e.g. [4], [5], [6] or [7]). Another approach is to give graphical support, like e.g. [8] or [9] with icon resp. block-based programming approaches.

Since the creation of skill-based robot programs still requires expert knowledge of the DSL, and parameters like compliances or other controller gains are hard to tune, different approaches to make skill-based programming more user-friendly have been exploited. For example, Lämmle et al. applied deep reinforcement learning (DRL) to automatically create a robot program from learning in a physical simulation environment [10]. Bargmann et al. presented a programming-by-demonstration approach based on the iTaSC formalism, using external measurements only and addressing the difficulties of combined position and force control [11].

B. Mixed Reality Assisted Programming

While augmented reality has been used for more than a decade in the context of robot programming, e.g. by augmenting camera images in a 2D desktop interface [12], a tracked pen for trajectory definition with overlay projection [13] or a head-mounted device [14], with a haptic input device [15] or with gesture detection [16], the use of MR has recently increased with the release of the Microsoft HoloLens 1 and 2 as key enabler.

Quintero et al. implemented an MR user interface that allowed the user to create trajectories in free space or on contact surfaces, to visualize the trajectories and to online reprogram during simulation and execution [17]. They found less teaching time and better performance contrasting a higher mental workload compared to a kinesthetic teaching.

Ostanin et al. created a hololens user interface for trajectory creation in simulation or on the real robot [18]. They used a simple set of buttons to choose from five simple robot actions and a three-dimensional interactive marker to specify and manipulate the trajectory points. An obstacle-avoiding path planning algorithm was used to enhance the user experience.

Gadre et al. implemented a hololens user interface for robot task creation. The user could choose from a list to add robot actions like waypoints, to create trajectories, or to visualize and execute motions [19]. Their study on two pick-and-place tasks found that the MR interface required less work, was easier to use, and more natural compared to a 2D interface.

So, while MR has been used to program industrial robots, the existing works focus on trajectory teaching and pick-and-place tasks. There exists no combination of skill-based programming and MR yet.

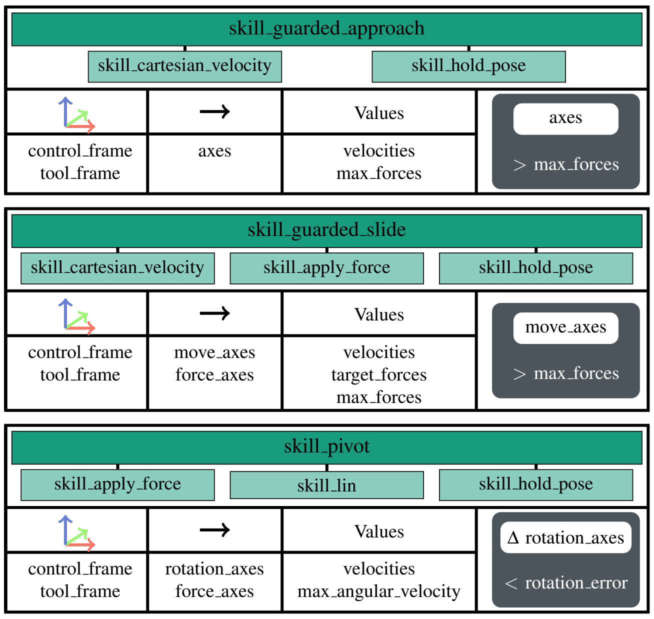


Fig. 2: Composition, parameters and monitors of three complex pitasc skills.

III. Skill-based Programming with pitasc

As a comprehensive underlying skill model and library for robot programming, we use the pitasc skill model [20]. From the proposed skill library, we choose three exemplary complex skills for our visualization, which have in practice been proven to be sufficient for many assembly tasks [21].

A. Pitasc Skill Model

Pitasc is driven by three elementary types of processes: position-, velocity-, or force-controlled motions (each linear and angular). In the pitasc skill library, the skill *skill_lin* implements a (Cartesian) position-controlled motion, the skill *skill_cartesian_velocity* a velocity-controlled motion, and the skill *skill_apply_force* a force-controlled one. Their parameters *frame* and *coordinates* determine which axis of which coordinate frame is controlled.

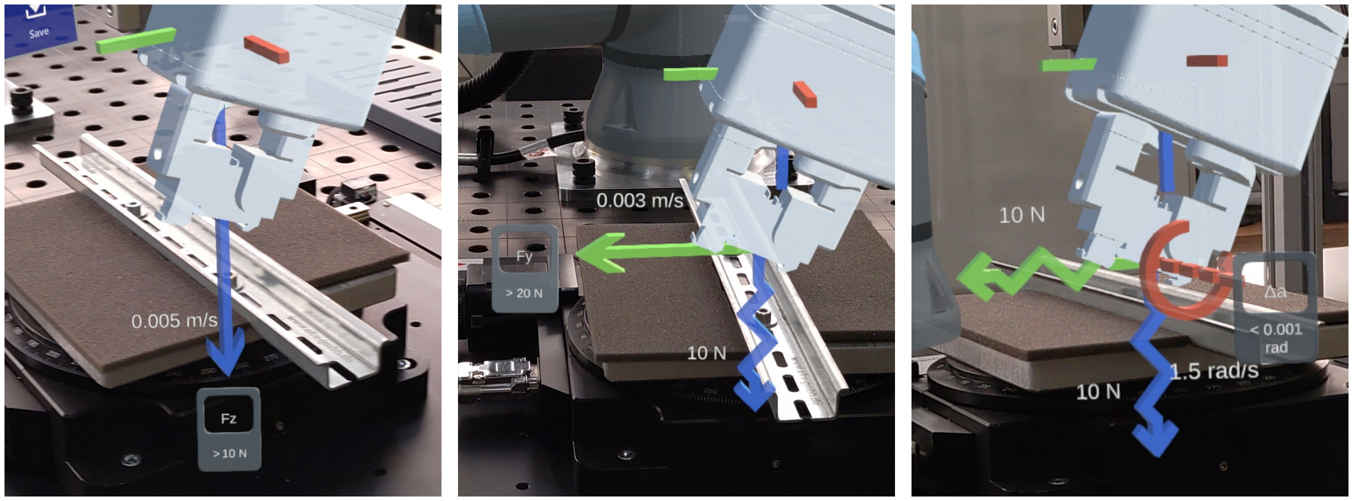
There are two container skills to compose these skills: the skill *skill_sequence* executes all its subskills sequentially, while *skill_concurrency* executes them concurrently. By putting, e.g., a *skill_cartesian_velocity* for the controlled frame's x- or y-coordinate and a *skill_apply_force* for the z-axis into a *skill_concurrency*, one can compose high-level skills – in this example a sliding motion.

To define transitions between skills, the pitasc model introduces *monitors* attached to the skills, observing, for example, force thresholds or position limits.

B. Selected Complex Skills

The compositions of the selected skills are shown in Fig. 2.

1) skill_guarded_approach: This skill moves towards a surface until a force contact is established. It consists of a *skill_concurrency* with the subskills *skill_cartesian_velocity*, which moves the tool frame along the specified *axes* of the *control_frame* with a given *velocity*, and a *skill_hold_pose*,



(a) skill_guarded_approach (b) skill_guarded_slide (c) skill_pivot

Fig. 3: Visualization of skill parameters: straight arrows (a, b) indicate linear velocity-controlled axes, the jagged arrows (b, c) indicate force-controlled axes and the rotation symbol (c) visualizes rotation around this axis. The respective values are displayed next to the respective axis. The monitor symbols visualize the control flow transitions at the corresponding axis by showing the observed parameter and the transition condition.

which holds the remaining DOFs in place. In addition, the member *max_forces* allows specifying a maximum force in the direction of motion; if this maximum force is exceeded, the next skill at the superior level is executed.

2) skill_guarded_slide: This skill slides along a surface and pushes perpendicular to the surface with a specific force until a force threshold in the direction of motion is exceeded. It consists of a *skill_concurrency* with three subskills: a *skill_cartesian_velocity*, which moves the tool frame in the specified *move_axes* of the *control_frame* with a given *velocity*, a *skill_apply_force* to apply a force along the specified *force_axes* (typically the tool's z-axis, pointing in the direction of the surface), and a *skill_hold_pose* to hold all remaining DOFs in place. As for the *skill_guarded_approach*, the member *max_forces* allows specifying a maximum force in the direction of motion.

3) skill_pivot: This skill inserts a part while keeping a contact. It consists of a *skill_concurrency* with the subskills *skill_apply_force*, which pushes the tool frame along the specified *force_axes* of the *control_frame*, a *skill_lin*, which aligns the specified *tool_frame* and the *control_frame* along the *rotation_axes* until the difference is below the specified *rotation_error*, and a *skill_hold_pose*, which holds all remaining DOFs in place. This skill is particularly useful when performing snapping motions, which turn a part while pushing or pulling in a perpendicular direction.

IV. Visualization and Interaction in Mixed Reality

To enable the user to create a robot program using the skills discussed in Sec. III-B, we visualize the parameters in the augmented world (Sec. IV-A) and provide a user interface in mixed reality to adapt the parameters to the user's needs (Sec. IV-B). During this teaching process, the user is always able to simulate the program (Sec. IV-C) or execute it on the real robot.

A. Parameter Visualization

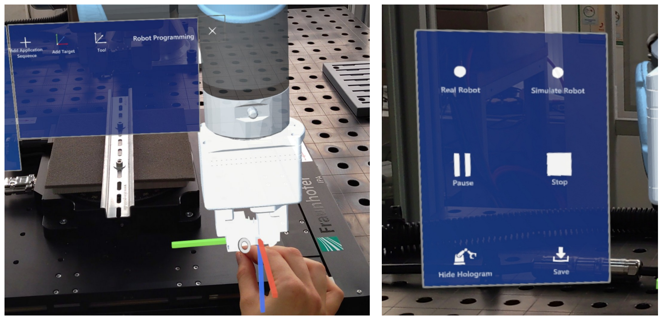
The selected complex skills (see Sec. III-B) have the following principal types of parameters:

- frames (*tool_frame*, *control_frame*)
- axes (*move_axes*, *force_axes*)
- numerical values (*velocities*, *target_forces*)
- control flow transitions (*max_forces*, *rotation_error*)

For frames, we use the popular concept of three RGB-colored axes for visualization, together with a text field that shows the corresponding frame's name. For axes, showing all of them with arrows would confuse the user in case of complex skills like *skill_pivot*, which pushes along two axes and moves (rotates) along another. Therefore, we distinguish between the type of motion. For linear motion axes, we use a simple arrow along with a text field showing the specified velocity. This arrow starts at the selected *tool_frame* and points in the direction of the chosen *control_frame* (see Fig. 3a, b); its color is chosen accordingly to its type (xyz – RGB). For rotational axes, we use a circular arrow around the rotation axes (see Fig. 3c).

For force-controlled axes, we use a different type of arrow inspired by the symbol of a mechanical spring (see Fig. 3b, c). To ensure that the text fields are always visible to the user, they are oriented toward the user via a small script that calculates the position orthogonal to the arrow (from the user's perspective).

For numerical values, we use a text field for simplicity. Sliders need a definition for the range, and setting them to an exact value in MR turned out to be difficult for the user, while the MR keyboard showed better usability. All values



(a) Frame teaching (b) Execution panel

Fig. 4: (a) Frame teaching by dragging the robot hologram's end-effector to the desired pose; (b) execution panel to validate the current state of the program on the digital twin (simulated robot) or on the real robot. 'Save' allows exporting the application to a file.

are visualized as text boxes directly at the corresponding axes.

To visualize the transitions between skills, a monitor symbol is shown at the end of the axis it observes. It shows the respective type of parameter, which can be a force (e.g. Fig. 3a: Fz) or a distance between axes (Fig. 3c: Δa, where the letter a means a rotation around the x-axis), and shows the transition condition below. Once the condition is met, the next skill is executed.

B. *User Interaction*

For the menu panels, there are in general two possibilities in MR: a head-up-display (HUD), which moves along with the user's head, or spatially anchored panels, which the user can move with hand gestures. Since on HUDs, the displayed information always covers a part of the field of view (FOV), and the FOV of current state-of-the-art MR-glasses is still smaller than a human's FOV, we choose spatially anchored panels as menu type. Another benefit is that when working with industrial robots, one can place the panels, e.g., on the walls of the robot cell, which provides an excellent overview and keeps the workspace free.

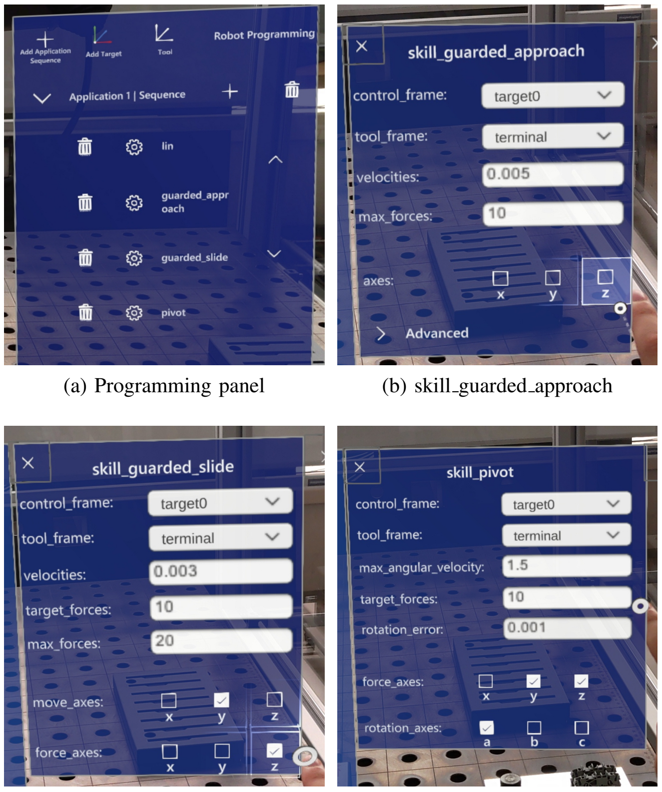
The presented application has different types of panels:

- a setup panel,
- a programming panel to actually create a robot program,
- a skill parameter panel to change skill parameters,
- an execution panel to run the program in simulation or on the real robot.

The robot programming panel is shown in Fig. 5a. At the beginning, the user sees an empty list, and by clicking the 'add' button, skills of the skill library can be added.

One click on the skill opens the skill parameter panel (see Fig. 5b-d) next to the application panel, where the user can set the skill parameters of the selected skill. These parameters are directly visualized at the tool center point of the robot, transformed from a text into a 3D experienceable visualization (see Fig. 3).

For teaching new frames, the user can drag the end-effector to its desired position and orientation (see Fig. 4a).



skill_guarded_approach
control_frame: target0
tool_frame: terminal
velocities: 0.005
max_forces: 10
axes: x [ ] y [ ] z [ ]

skill_guarded_slide
control_frame: target0
tool_frame: terminal
velocities: 0.003
target_forces: 10
max_forces: 20
move_axes: x [ ] y [✓] z [ ]
force_axes: x [ ] y [ ] z [✓]

skill_pivot
control_frame: target0
tool_frame: terminal
max_angular_velocity: 1.5
target_forces: 10
rotation_error: 0.001
force_axes: x [ ] y [✓] z [✓]
rotation_axes: a [✓] b [ ] c [ ]

(a) Programming panel (b) skill_guarded_approach

(c) skill_guarded_slide (d) skill_pivot

Fig. 5: Programming panel (a) and skill parameter panels (b-d) to set the parameters of the respective skill. Additional expert parameters, e.g. controller gains, are provided by an 'advanced' tab (b).

Clicking on 'Add Target' creates a frame at the current pose, which can then be named and used as a skill parameter.

The execution panel (see Fig. 4b) allows to simulate the application, which means executing the pitasc program on the digital twin, a digital representation of the robot and the environment, which is visualized as a holographic robot within the real world (see Fig. 1). The program can also be executed on the real robot. The pause/resume and stop buttons help to inspect certain moments of the program in detail. By clicking 'Save', the application is exported in a pitasc DSL (XML) file, which can be executed independently from our application using the pitasc executor [22].

C. *Digital Twin and Force Simulation*

To allow a safe and effortless detection of wrong parametrizations or skill orders, a digital twin of the robot along with contact force simulation is used to simulate the robot program for the user.

The digital twin requires a definition of the real robot, its end-effector, and the part of the environment the robot shall interact with. With these, a digital twin can automatically be created without any configuration overhead. The physical simulation calculates the contact forces between the simulated robot and the virtual environment, acting as a virtual force-torque sensor.

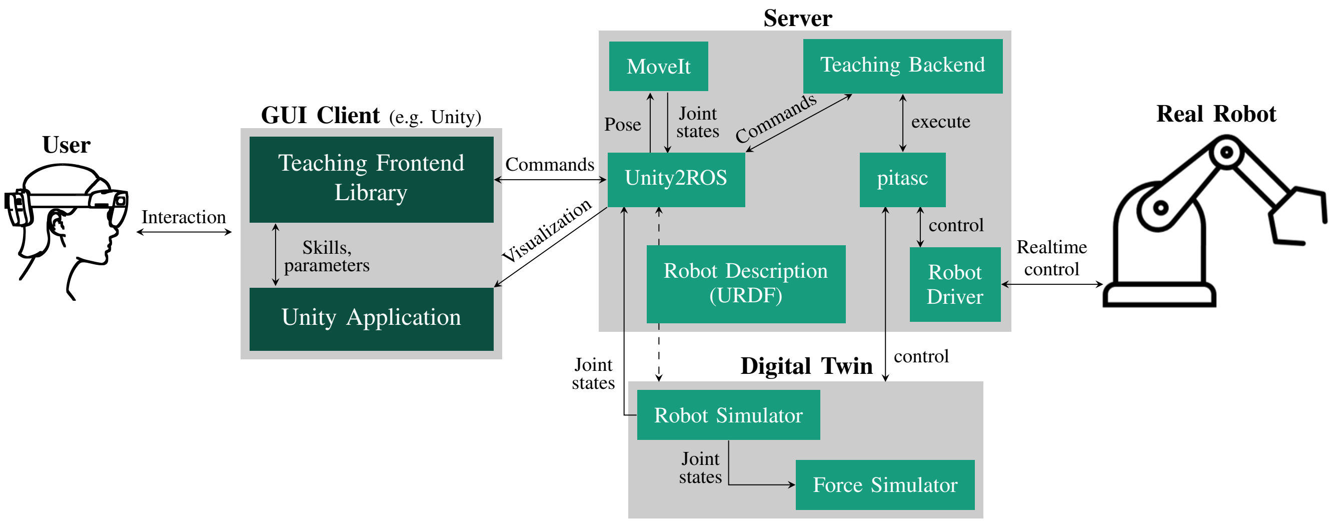


Fig. 6: Proposed software architecture: The user programs a robot application using the HoloLens GUI client, which communicates with the server via a Unity2ROS bridge. The teaching backend holds all information about the current program and allows execution on the digital twin or the real robot via pitasc. The digital twin is derived from the robot description and consists of a robot simulator and a force simulation. MoveIt [25] is used for coordinate frame teaching, calculating the inverse kinematics for a given pose.

Seeing robot programming as a reinforcement learning process, the human here acts as the RL-agent, exploring different sets of skills and their parameters and evaluating them in a virtual environment. Once a satisfactory state has been achieved, the user can execute it on the real robot.

V. Architecture and Implementation

We introduce a client-server-based architecture for robot programming. This is especially required in industrial environments to ensure that the program is not lost once the MR glasses' battery is empty or if one wants to switch to text-based programming at some point. Fig. 6 shows the main components and the communication between them.

A. Server

The teaching backend node holds the current program and model information, providing a GraphQL-Interface [23] to access or modify data. GraphQL is an alternative to a REST API, using json-based queries and thus being independent of the communication protocol.

Other server-side components, which all communicate via Robot Operating System (ROS) [24], are the descriptions of the physical robot and the environment using the Unified Robot Description Format (URDF), a robot simulator and a force simulator to simulate force-controlled applications, the driver of the real robot, and a pitasc node to execute pitasc applications. Since the interface of pitasc to the robot is based on ros_control, in general, all robots with a ROS driver and a VelocityJointInterface can be controlled by pitasc and be taught by our application.

For the frame teaching (by dragging the hologram's end-effector, see Sec. IV-B), the inverse kinematics are calculated with the MoveIt Motion Planning Framework [25].

B. GUI Client

We use a Microsoft HoloLens 2 as MR glasses. The application is developed with Unity3D [26] and MixedRealityToolkit [27]. UnityRoboticsHub [28] is used to load the robot model and to communicate with ROS. A custom frontend library takes care of the GraphQL communication with the teaching backend, providing access to the pitasc application and execution. This decoupling allows reusing the frontend library for different kinds of clients.

C. Digital Twin

For simulating the contact forces, the physics engine MuJoCo [29] is used. We implemented an interface to construct the environment from the URDF description of the real robot environment and feed the end-effector poses from the robot simulator into the engine.

Therefore, we do not rely on correct inertial data of the robot links in the robot URDF description (which, in our experience, are rarely accurate), since we do not physically simulate the robot joints and its controller. Instead, the robot simulator assumes that the robot controller follows the given joint velocities with a delay of three time steps and a Gaussian position noise of 1×10^{-5} rad (standard deviation). MuJuCo then calculates the forces acting on the end-effector resulting from this motion.

To smooth the coupling of the robot and force simulations and properly utilize MuJoCo's soft constraint model, we insert a coupled compliant element (stiffness $\approx 232\,\mathrm{kN/m}$ $(\mathrm{kNm/rad})$, damping $\approx 1000\,\mathrm{Ns/m}$ $(\mathrm{Nms/rad})$) between the tool center point of the robot and the end-effector in the force simulation. While the robot pose updates with 125 Hz, MuJoCo runs with 33 kHz to simulate the involved dynamics sufficiently.

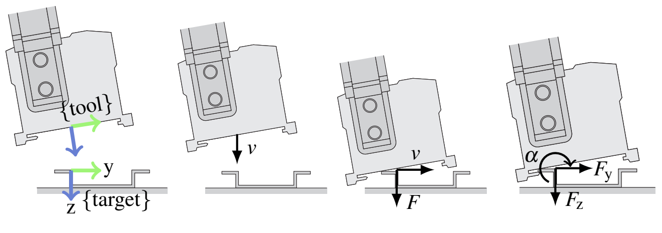


Fig. 7: Typical steps during the assembly of a terminal on a top hat rail [20]. v means a velocity, F a force, and α a rotation.

VI. Demonstration and Discussion

We validate our solution on a top hat rail mounting task (see Fig. 7). Fig. 1 shows the experimental setup: The rail is mounted on a positioning table, which is turned off and not part of the experiment. The foam beside the rail reduces light reflections. We use a Universal Robots UR10e and its integrated force-torque sensor.

The teaching of picking the part is omitted here since we focus on the force controlled assembly part, and part picking consists of just a few transfer motions (*skill_lin*) and a skill that closes the gripper.

The terminal can be mounted using the following skills:

- a *skill_lin* to a start position with a tilting angle of approx. 20° (see Fig. 7a)
- a *skill_guarded_approach* to move towards the rail until a certain force is measured (see Fig. 7b)
- a *skill_guarded_slide* to keep contact to the rail while moving to the side until the snap fits in the rail and causes a force (see Fig. 7c)
- a *skill_pivot* to rotate to the target orientation while keeping contact to the rail (see Fig. 7d).

The robot programming took approx. 10 minutes using the mixed reality interface, including frequent simulations to check the validity of the program. To validate the architecture, we implemented an rviz-plugin [30] as another GUI client, and successfully created the same program, completely virtually, without any server-side adaptations.

After achieving a satisfying execution on the digital twin, we executed the program on the real robot using the execution panel (see Fig. 4b), which already succeeded on the first try without any adaptions or tuning.

Fig. 8 compares the simulated forces and the measured forces of the real robot for two different parameter sets, showing a qualitatively and quantitatively satisfying correspondence with root mean square errors in z-direction of 1.47 N (F_h) and 1.48 N (F_l), respectively. The deviation in the y-force during *skill_guarded_slide* is caused by different friction coefficients in the real world and simulation and can trigger the axis monitor of *skill_guarded_slide*. Thus, the *max_forces* parameter must not be chosen too low – this applies to both the simulation and the real robot. Since the simulation mainly helps to (qualitatively) visualize force-controlled skills, the deviation is acceptable but can be improved by more accurate modeling and parameter tuning.

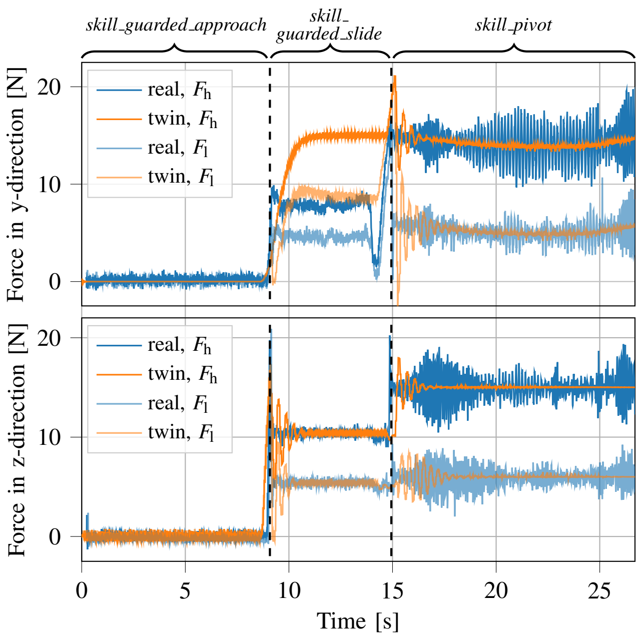


Fig. 8: Comparison of real and simulated forces in y- and z-axes of {target} frame (see Fig. 7a for axes definition) for a parameter set with high and one with low *target_forces* (F_h = {10 N, 15 N} and F_l = {5 N, 5 N}, *target_forces* in z).

While menus and buttons showed good usability in MR, the keyboard interaction was troublesome and error-prone. One overcome could be a multimodal input approach, with MR and classical input devices like mouse and keyboard, beside existing deep learning approaches for setting parameters [10]. Furthermore, the interface needs an abstraction mechanism for encapsulating skills in a high-level skill (like a possible *skill_mount_terminal*). This allows for simple reuse and reducing the number of parameters.

VII. Conclusion and Outlook

We presented a tangible MR-based robot teaching approach for skill-based programming. By designing visualizations for different types of parameters and using them in the user interface, we demonstrated how skill-based programming can profit from the 3D-visualization capabilities of MR. Ad-hoc validation on a digital twin, including contact force simulation, reduces errors in programming, avoiding potential hardware damages, and lessens the gap between simulation and reality. All in all, we gave an impression of how a future robot teaching panel and teaching process could look like.

In the future, we will work on a generic visualization model for all types of parameters and skills of the skill library. Furthermore, we will evaluate the user experience in user studies to get quantitative information about the practical usefulness of MR in skill-based programming. We plan a comprehensive user study with more than 20 participants where we aim to compare different types of programming interfaces along with different levels of experiences of the participants.

REFERENCES

[1] International Federation of Robotics, "Executive summary world robotics 2021 industrial robots," *World Robotics Report*, pp. 12–16, 2021.

[2] Verein Deutscher Ingenieure, *Montage- und Handhabungstechnik, Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole*. Berlin, 1990.

[3] 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.

[4] M. Klotzbücher, R. Smits, H. Bruyninckx, and J. De Schutter, "Reusable hybrid force-velocity controlled motion specifications with executable domain specific languages," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 4684–4689.

[5] D. Vanthienen, M. Klotzbu, J. De Schutter, T. De Laet, H. Bruyninckx, *et al.*, "Rapid application development of constrained-based task modelling and execution using domain specific languages," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 1860–1866.

[6] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using uml/p statecharts," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 461–466.

[7] F. Nägele, L. Halt, P. Tenbrock, and A. Pott, "A prototype-based skill model for specifying robotic assembly tasks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 558–565.

[8] R. Bischoff, A. Kazi, and M. Seyfarth, "The morpha style guide for icon-based programming," in *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication*, 2002, pp. 482–487.

[9] N. Müller, M. Wagner, and P. Heß, "Intuitive Programmierung für die Mensch-Roboter-Kollaboration," *Zeitschrift für wirtschaftlichen Fabrikbetrieb*, vol. 112, no. 7-8, pp. 477–480, 2017.

[10] A. Lämmle, T. König, M. El-Shamouty, and M. F. Huber, "Skill-based programming of force-controlled assembly tasks using deep reinforcement learning," *Procedia CIRP*, vol. 93, pp. 1061–1066, 2020.

[11] D. Bargmann, P. Tenbrock, L. Halt, F. Nägele, W. Kraus, and M. F. Huber, "Unobstructed programming-by-demonstration for force-based assembly utilizing external force-torque sensors," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 119–124.

[12] B. Akan, A. Ameri, B. Cürüklü, and L. Asplund, "Intuitive industrial robot programming through incremental multimodal language and augmented reality," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3934–3939.

[13] A. Gaschler, M. Springer, M. Rickert, and A. Knoll, "Intuitive robot tasks with augmented reality and virtual obstacles," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6026–6031.

[14] S.-K. Ong, A. Yew, N. K. Thanigaivel, and A. Y. Nee, "Augmented reality-assisted robot programming system for industrial applications," *Robotics and Computer-Integrated Manufacturing*, vol. 61, p. 101820, 2020.

[15] D. Ni, A. Yew, S. Ong, and A. Nee, "Haptic and visual augmented reality interface for programming welding robots," *Advances in Manufacturing*, vol. 5, no. 3, pp. 191–198, 2017.

[16] J. Lambrecht and J. Krüger, "Spatial programming for industrial robots based on gestures and augmented reality," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 466–472.

[17] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. M. Van der Loos, and E. Croft, "Robot programming through augmented trajectories in augmented reality," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1838–1844.

[18] M. Ostanin, S. Mikhel, A. Evlampiev, V. Skvortsova, and A. Klimchik, "Human-robot interaction for robotic manipulator programming in mixed reality," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 2805–2811.

[19] S. Y. Gadre, E. Rosen, G. Chien, E. Phillips, S. Tellex, and G. Konidaris, "End-user robot programming using mixed reality," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 2707–2713.

[20] F. Nägele, "Prototypbasiertes Skill-Modell zur Programmierung von Robotern für kraftgeregelte Montageprozesse," dissertation, University of Stuttgart, 2021.

[21] L. Halt, F. Nägele, P. Tenbrock, and A. Pott, "Intuitive constraint-based robot programming for robotic assembly tasks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 520–526.

[22] Fraunhofer IPA, "pitasc modular system." [Online]. Available: https://www.pitasc.fraunhofer.de/en.html

[23] GraphQL Foundation. Graphql — a query language for your api. [Online]. Available: https://graphql.org/

[24] M. 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*, Kobe, Japan, May 2009.

[25] PickNik Robotics, "MoveIt Motion Planning Framework." [Online]. Available: https://moveit.ros.org/

[26] Unity Technologies, "Unity Real-Time Development Platform." [Online]. Available: https://unity.com/

[27] Microsoft, "MixedRealityToolkit." [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05

[28] Unity Technologies, "MixedRealityToolkit." [Online]. Available: https://github.com/Unity-Technologies/Unity-Robotics-Hub

[29] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[30] Open Source Robotics Foundation, "rviz - ROS Wiki." [Online]. Available: http://wiki.ros.org/rviz