Hiding task-oriented programming complexity: an industrial case study

Enrico Villagrossi[a], Michele Delledonne[a,b], Marco Faroni[a], Manuel Beschi[a,b], and Nicola Pedrocchi[a]

[a]Institute of Intelligent Industrial Technologies and Systems for Advanced Manufacturing, National Research Council of Italy, Via A. Corti 12, 20133, Milan, Italy;
[b]Department of Mechanical and Industrial Engineering, University of Brescia, Via Branze 39, 25123 Brescia, Italy

ABSTRACT
The ease of use of robot programming interfaces represents a barrier to robot adoption in several manufacturing sectors because of the need for more expertise from the end-users. Current robot programming methods are mostly the past heritage, with robot programmers reluctant to adopt new programming paradigms. This work aims to evaluate the impact on non-expert users of introducing a new *task-oriented* programming interface that hides the complexity of a programming framework based on ROS. The paper compares the programming performance of such an interface with a classic *robot-oriented* programming method based on a state-of-the-art robot teach pendant. An experimental campaign involved 22 non-expert users working on the programming of two industrial tasks. *Task-oriented* and *robot-oriented* programming showed comparable learning time, programming time and the number of questions raised during the programming phases, highlighting the possibility of a smooth introduction to *task-oriented* programming even to non-expert users.

1. Introduction

1.1. *Context*

Industrial robots in manufacturing have steadily grown in the last few decades. According to the World Robotics Industrial Robot report 2021 (International Federation of Robotics, 2022), two of the most addressed applications of industrial robots are material handling and components assembly, which are mainly pick&place tasks. The structure of a pick&place program is frequently defined as a long list of *"move to"* instructions.

Since early robotic applications, the definition of the robot program was based on a *teaching-by-showing* approach, where the programmer guides the robot manually along the desired trajectory (*lead-through* programming), or with the Teach Pendant (*drive-through* programming). The use of *lead-through* programming is frequent for collaborative robots. The lightweight structure and the low payload allow for dragging the end-effector intuitively across the workspace, while the design of the robot allows for a safe interaction following the current safety standards as the ISO/TS 15066:2016 (ISO/TC 299 Robotics, 2016).

The evolution of robot programming techniques brought *robot-oriented* programming languages (Yang et al., 2015). They are high-level programming languages, primarily based on BASIC and PASCAL, such as the ABB Rapid, the Fanuc Karel and the Kuka KRL. Such programming languages are integrated with advanced robotic functions that allow the development of complex robotic applications but retain the *teaching-by-showing* programming mode. However, the languages continued to develop, gradually incorporating features from the rest of the programming world. As a drawback, each robot manufacturer developed its proprietary *robot-oriented* programming language incompatible with the others. Reaching an essential knowledge of proprietary languages requires a reduced amount of time. However, a deep knowledge of a robotic platform requires great experience and training, which is why robotic programmers tend to become specialised in a few specific robot brands. This approach may cause resistance by robot programmers and robotic system integrators to acquire new programming skills, learn different programming paradigms, and use different robot brands.

The *robot-oriented* programming languages are currently the most widespread for industrial applications, even if offline programming environments more and more often flank them. Thanks to accurate robotic cell modelling, the development and the simulation of the robot programs, with offline tools, before on-site testing allows saving time (Pan et al., 2012). Integrating offline programming environments with CAD systems has also led to the automatic generation of the robot part program (Castro et al., 2019). Classic use is for continuous processes such as machining, painting and arc welding applications (*i.e.,* a vast number of via-points are generated by a CAD/CAM system and interpolated by the robot). Offline programming can partially generate collision-free trajectories for a given planning environment. More frequently, they are used to check and highlight the presence of collisions between the robot and the environment. As for *robot-oriented* programming languages, every robot manufacturer developed its offline programming environment, such as ABB RobotStudio, Kuka.Sim, Motoman MotSim, Fanuc RoboGuide. Nevertheless, an accurate 3D model of the cell is not always available, and the construction of the cell can bring inaccuracies w.r.t. the original design. Hence, adopting the traditional *drive-through* programming is frequent during the commissioning phase to re-teach trajectories via-points.

Despite (i) the increasing complexity of robotic applications, (ii) the evolution of *robot-oriented* programming languages, (iii) the adoption of offline programming tools, the approach to robot programming and how the robot programs are structured have barely changed over the years. The robot program maintains a rigid structure as a list of instructions coded and saved into the robot memory. Moreover, *robot-oriented* programming languages and offline programming tools are not easily handled by end-users unfamiliar with robotic knowledge.

The introduction of the ROS framework in robotic research has boosted the search for a programming approach oriented to standardisation enabling the technological transfer of state-of-the-art algorithms from research to industry. ROS adoption can

bring several innovations, such as *task-oriented* programming, state-of-the-art motion planners, dynamic motion planning, simulation environments, and easy sensor integration. Unfortunately, ROS does not penetrate the world of industrial applications despite the efforts made by the ROS-industrial consortium (ROS-Industrial consortium, 2022). Currently, ROS-industrial is moving on ROS2 with a new tentative to relaunch the ROS initiative dedicated to industrial purposes. The main barrier to ROS adoption in the industry remains the slow learning rate for the robotic technicians unfamiliar with General Purpose Languages (GPL), such as *C/C++*, *Java* and *Python*, and the concepts of a robotic framework. Improvements to programming interfaces are required to attract new users unfamiliar with GPL bringing together the advanced features provided by ROS with the ease of use of classical *robot-oriented* programming languages and a design that enables *task-oriented* programming.

1.2. *Motivation and Contribution*

Despite several demonstrations in robotic research, the penetration of advanced robot programming techniques, such as visual programming or programming by demonstration, is facing barriers in the industrial context (Villani et al., 2018). The main obstacles are the robustness of the advanced programming algorithms, the complexity of the programming interfaces and the technical heritage of robot programmers reluctant to adopt new programming techniques because of their familiarity with textual *robot-oriented* programming languages.

The Manipulation Framework was introduced (MF), as described in (Villagrossi, Pedrocchi, and Beschi, 2021), to evolve the current industrial robot programming paradigm moving from a *robot-oriented* paradigm to a *task-oriented* programming tool, to reduce the programming time and to simplify the development of complex collaborative applications. The MF improved with the introduction of a *task-oriented* Graphical User Interface (MFI hereafter) designed to hide the complexity of the framework based on ROS. This paper aims to compare the acceptance level, the ease of use and the effectiveness of the *task-oriented* interface to program industrial tasks by nonexpert robot programmers and end-users. The paper compares the tasks programming made with the MFI and a classic *lead-through* programming approach made with the robot Teach Pendant (TP). The setup used for the test was a UR10e robot, with its TP, which is nowadays considered the state-of-the-art of robot TPs in terms of intuitiveness and ease of use[1].

The study involved a heterogeneous group of university students from multiple STEM faculties and machine tool operators (with different backgrounds) as end-users testers. Only a few testers had little prior experience in robot programming, and no one had seen either the MFI or the TP used for the experiments.

The experimental results demonstrate that using a *task-oriented* framework, as the MFI, can introduce slightly longer learning time but can bring several benefits w.r.t. a standard programming technique. Advantages are evident when it is necessary to deal with: task repetition, robot reprogramming and collision-free motion planning. The parameters monitored during the experimental tests to compare the interfaces are the learning time, the programming time, the number of questions made during the programming, the testing time and the reprogramming time.

The comparison between the two programming methods, the results of an experimental campaign involving industrial tasks in a real industrial environment and the

---

[1]Standard industrial robot TPs are, in general, much more complex than UR's TP.

presence of machine tools operators in the study (which is not common in previous works) constitute interesting points of novelty.

1.2.1. Paper outline

The paper is organised as follows: Section 2 analyses the related works, Section 3 describes the experimental setup, the experiments design, the programming interfaces and the method. Section 4 discusses the results obtained during the experimental campaign. Finally, Section 5 reports the conclusions and the future works.

2. Related works

Programming interfaces evolved with the improvements in the technology (Siciliano et al., 2010) providing a wide range of solutions (Tsarouchi, Makris, and Chryssolouris, 2016; Mukherjee et al., 2022; Heimann and Guhl, 2020). However, a gap remains to bring robot programming closer to end-users (Ajaykumar, Stiber, and Huang, 2021), such as production operators, and promote the spread of robotic applications to new manufacturing fields (Ajaykumar, Steele, and Huang, 2021). Advanced approaches, such as programming by demonstration, visual programming, augmented/virtual reality, and natural language programming, are characterised by high intuitiveness since they constitute instances of natural and tangible user interfaces (NUIs and TUIs).

Programming by demonstration is a technique where the programmer can demonstrate the task to the robot (Billard et al., 2008; Zhang, Wang, and Xiong, 2016). A common approach to programming by demonstration is the evolution of *lead-through* programming method exploiting admittance/impedance control algorithms where the user physically interacts with the robot through robot manual guidance (Mohammad Safeea, 2022). The most common approach uses force/torque sensors mounted between the robot flange and the end-effector (Bascetta et al., 2013). Conversely, modern collaborative robots, such as the Kuka LBR or the Franka Emika, integrate torque sensors into robot joints. The drawbacks of admittance/impedance control algorithms are accurate parameters tuning, which can bring stability issues (Ferraguti et al., 2019), additional sensing (force/torque sensors), and they are not available as a software feature for most industrial robots. Moreover, the demonstration accuracy is not enough for most industrial applications.

Alternative methods, less common, provide the use of voice, vision (Lu, Berger, and Schilp, 2022) and motion capturing (Makris et al., 2014).

Visual programming makes programming more approachable for non-experts (Coronado et al., 2020; Huang et al., 2020). In literature, visual programming interfaces commonly use flow diagrams, behaviour trees, blocks, and icons (Stenmark, Haaae, and Topp, 2017). Some visual programming systems propose personal graphical user interfaces (Schou et al., 2018) that use buttons, menus, windows, textual inputs, and sliders. The standard IEC-61131-3, which defines programming languages for automation, provide visual programming languages, such as the Function Block (FB) or the Sequential Flow Chart (SFC), that can also be used for robot programming as standard languages (Thormann and Winkler, 2021; Rendiniello et al., 2020). These languages are suitable for beginner programmers. The execution speed of visual applications is slow, and the programming requires more time than a textual one. A complex task requires a large number of operations, users spend time making room for things, encasing and rearranging them in macros, and the overall program can get crowded.

Augmented/virtual reality allows overlapping the real-world environment with a virtual one. With this technology, some information or programming tools can appear directly in the environment. The possibility of overlapping a virtual robot and objects allows the operator to use the programming by demonstration without interacting directly with the robot (Blankemeyer et al., 2018). Visualising virtual panels with programming information (*e.g.,* robot trajectory or parameter values) allows the operator to make decisions (Gadre et al., 2019). Using physical auxiliary tools (Ong et al., 2020) or object detection software (Apostolopoulos et al., 2022) can facilitate the user programming. This technology presents a high implementation cost; it needs to be more flexible as it is programmed for a specific task and prone to failures in case of environmental changes.

Natural language programming uses speech and text to create programs. Usually, this technique works in parallel with another programming technique. The high complexity of human language requires some constraints, and this type of language cannot describe the action in its entirety. In literature, natural language programming was combined with programming by demonstration (Quintero et al., 2018), visual programming (Huang and Cakmak, 2017), and augmented reality (Andronas et al., 2021). The use of speech recognition is hardly usable in industrial environments; the high noise that characterises these working places makes it difficult to recognise voice commands.

Tangible programming is a technique that uses physical objects to define program structures. By positioning specific objects in the environments, it is possible to define the desired object for a particular action, the action to perform, and the areas where the actors perform it (Sefidgar, Agarwal, and Cakmak, 2017). Another approach is to use the specific card to define the task structure; the card type and order represent the action types and order (Kubota et al., 2020). The programming of a complex task might require a large number of objects making the environment chaotic. Some actions are difficult to describe through objects, especially if they need good accuracy.

Alternative ways to automate the creation of the robot part-program provide the direct use of a CAD model to extract the necessary information to control the robot, bypassing the robotic offline programming tools as in (Neto and Mendes, 2013). This approach is helpful for continuous processes.

The EU projects Robo-Partner (ROBO-PARTNER Consortium, Nov. 2013 - Apr. 2017), Sharework (Sharework Consortium, Nov. 2018 - Oct. 2022) and Sherlock (Sherlock Consortium, Oct. 2018 - Sep. 2022a), within their scopes, investigated the use of advanced programming techniques in industrial scenarios. Robo-Partner faced the programming of an assembly task exploiting programming by demonstration techniques through audio commands, visual programming, and direct robot arm manipulation by the user through force/tactile sensing, as described in (Michalos et al., 2014). The goal of Robo-Partner was to bring robotics closer to SMEs where poor robotic knowledge represents a barrier to robot adoption. Sharework aims to introduce dynamic task planning able to assign the robot actions to the robot controllers autonomously (Umbrico et al., 2022). The assignment evaluates the evolution of a collaborative task controlling the robot through ROS libraries (Faroni et al., 2020). The project developed robot motion planning algorithms based on learning by demonstration and provided intuitive human-robot interaction methods based on virtual reality. Sherlock aims to provide zero-programming robotics for collaborative and medium-high payload robots. The project developed a programming architecture that mixes augmented reality (*i.e.,* indirect interaction) and manual guidance (*i.e.,* direct interaction) based on force sensors. Mixed reality roughly defines the trajectory starting and ending point, while manual guidance can refine and adjust precisely the final position.
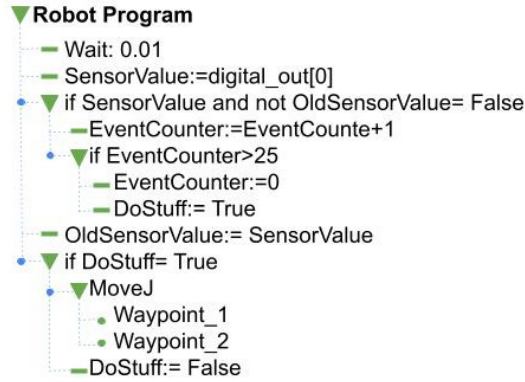
Figure 1. Robot-oriented program.

Using a motion planner and a digital twin enables the generations of collision-free trajectories (Sherlock Consortium, Oct. 2018 - Sep. 2022b).

Several frameworks such as: CORBA (Schmidt, Levine, and Mungee, 1998), YARP (Metta, Fitzpatrick, and Natale, 2006), ROS (Stanford Artificial Intelligence Laboratory et al., 2022), IMI2S (Anzalone et al., 2014) are currently used to control the robot. Adopting frameworks is a prerogative of robotic research; no relevant spread is currently perceptible in the industry. Among the others, the most successful is ROS, representing a standard *de facto* for robotic research programming. ROS brings advanced features and open-source packages that allow for a *task-oriented* approach. On the other hand, ROS imposes the use of GPL languages, which are often adopted only in research[2].

3. Materials and Methods

3.1. *The programming interfaces*

This paper compares a *robot-oriented* and a *task-oriented* programming interface. *Robot-oriented* programming focuses on primitive robot movements that the robot can perform. The user combines these primitive actions into a sequence to obtain the desired program. Figure 1 shows an example of a robot-oriented program.

Task-oriented programming focuses on the task. The user combines high-level actions by setting the parameters required by the process operation rather than the robot's motion. The user does not define the primitive action from scratch, as the framework programmer previously defined the task structure. The user codes in an intuitive language, as shown in Figure 2.

The experiments involved one programming interface for each type. The *robot-oriented* programming interface is the UR10e TP, and the *task-oriented* one is a novel GUI built on the top of the Manipulation Framework (Villagrossi, Pedrocchi, and Beschi, 2021).

---

[2]Kuka Sunrise.OS (Kuka AG, 2022), dedicated only to Kuka LBR iiwa, is currently a unique example of a robotic OS that provides libraries based on GPL languages, like Java, to program the robot.

```
PLACE BEARING1 SO (SHAFT FITS BEARING1.HOLE) AND
                  (BEARING1.BOTTOM AGAINST SHAFT.LIP)
PLACE SPACER SO (SHAFT FITS SPACER.HOLE) AND
                  (SPACER.BOTTOM AGAINST BEARING1.TOP)
PLACE BEARING SO (SHAFT FITS BEARING2.HOLE) AND
                  (BEARING2.BOTTOM AGAINST SPACER.TOP)
PLACE WASHER SO (SHAFT FITS WASHER.HOLE) AND
                  (WASHER.BOTTOM AGAINST BEARING2.TOP)
SCREW-IN NUT ON SHAFT TO (TORQUE = t0)
```
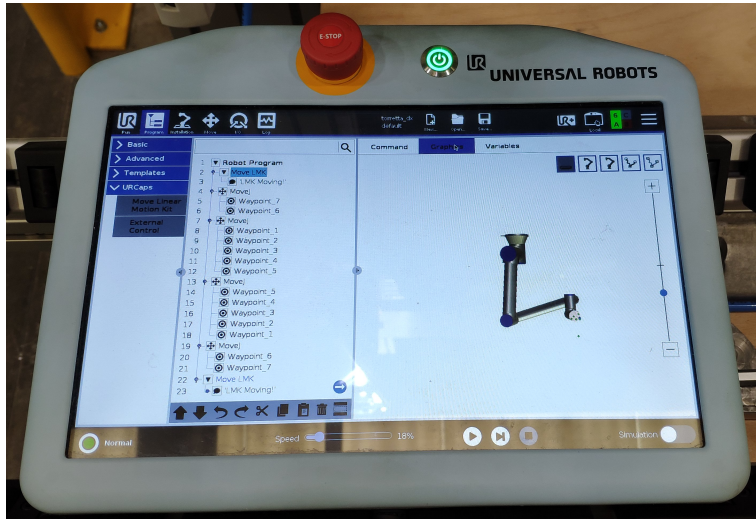
Figure 2. Task-oriented program.

Figure 3. UR10e teach pendant.

3.1.1. Robot-oriented programming interface: UR10e teach pendant interface

The UR10e TP is a highly intuitive programming interface based on a touch screen device without physical buttons (apart from the on/off and the emergency buttons); Figure 3 shows the UR10e TP. The TP enables the robot movements in jogging mode, the access to robot configurations and parameters, and the robot programming through a *robot-oriented* high-level programming language. A new robot program requires a sequence of *move* instructions by teaching the starting and the ending robot configuration to be interpolated. The teaching of robot position can be done by *lead-through* programming by moving the robot with the so-called manual guidance mode. The programmer must add intermediate robot configurations (via-points) to guarantee collision-free trajectories. The TP interface (TPI) provides specific functions to manage the gripper activation.

3.1.2. Task-oriented programming interface: manipulation framework interface

The MF is a software package dedicated to manipulation tasks (Villagrossi and Beschi, 2022a,b). MF provides some essential components to enable *task-oriented* robot programming. Figure 4 shows the structure of the MF. For a given manipulation task, the MF aims to: (i) process manipulation actions (such as pick, place, move to), relieving the programmer from the management and the execution of single actions; (ii) handle the kinematic model of the robotic system (arm + grasping system) being able to compute forward and inverse kinematics; (iii) embed motion planning functionalities to generate collision-free trajectories for a given planning scene and a given robotic system in the planning environment that can dynamically change; (iv) execute the
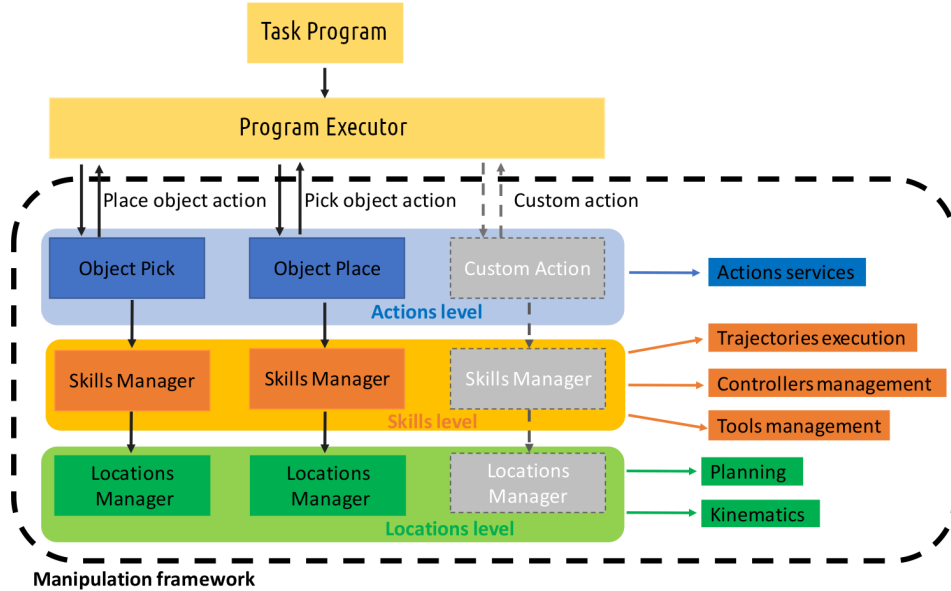
Figure 4. Manipulation framework layers description.

planned trajectories on the desired robotic system; (v) change the robot controller dynamically. The motion planning uses *MoveIt!* pipeline (Open source community, 2022). *MoveIt!* allows dynamic load motion planners as plugins, enabling collision-free motion planning for a given planning scene. Additional information is available in (Villagrossi, Pedrocchi, and Beschi, 2021).

MF's APIs require *C++* programming and ROS integration (*e.g.*, the creation of ROS nodes, the definition of configuration files, and dataset management.) To overcome this limit, a GUI was developed as MF Interface (MFI) (Delledonne and Beschi, 2022a,b). MFI has a structure designed for *task-oriented* programming. The interface allows to easily concatenate multiple *Actions* to assemble a *Task*. The user can set all the components using interaction with the robot being free to focus on the task definition. The user can choose between pre-programmed *Actions*, such as *Pick, Place, GoTo*. A program can be composed by dragging and dropping the desired *Actions*, as shown in Figure 5. A set of parameters defines each *Action*. For example, a *Pick* action requires the approaching pose, the grasping pose and the gripper closing force (see Figure 6b). Notice that the MFI does not require to specify via-points, as the MF will automatically plan the trajectories based on the task specifications.

A relevant feature of the MFI allows specifying an *Object Type* that groups several objects. The MF will automatically compute the poses for the object grasping (and IK solution) for every object of the same *Object Type* during the motion planning. The object with the optimal path planning will be selected.

Positioning the robot in the desired position is possible to acquire a robot pose, similar to classic TP programming. The MFI allows for both *lead-though* programming (if the robot has a force/torque sensor) and tele-operation, see Figure 7. The MFI can be easily connected to a vision system to automatically localise the objects' positions in the robot workspace, providing additional programming flexibility and autonomy.

The interface allows for *Action*, or sequence of *Actions* execution. The first step is the kinematics computation (*i.e.,* robot and objects involved in the task), consequently, the computation of the collision-free motion plans. An error message informs the operator
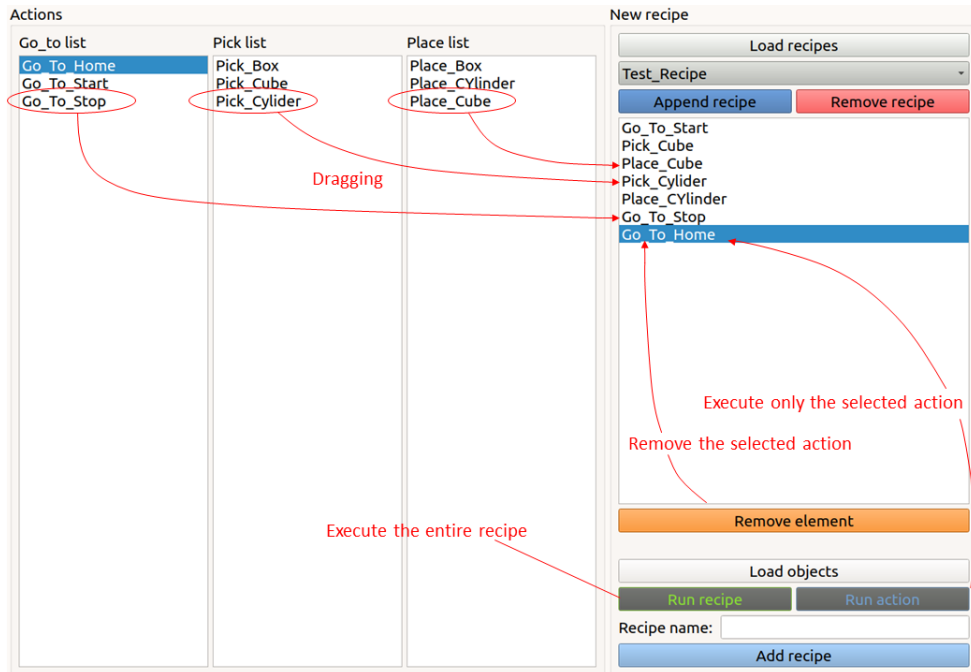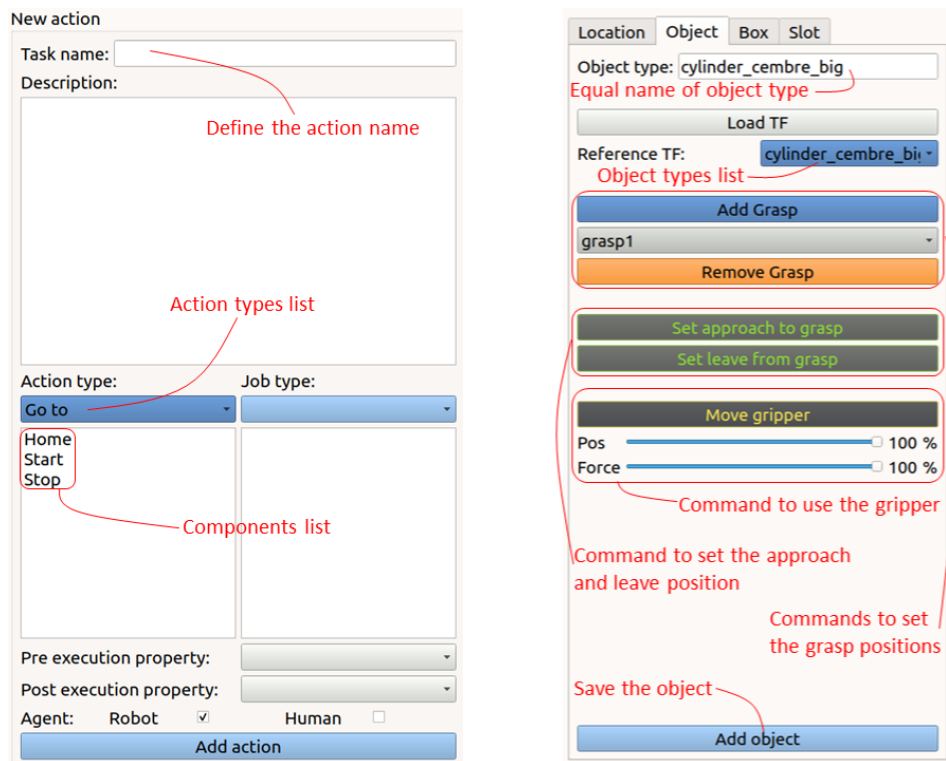
8

Figure 5. MFI: example of *Task* composition by concatenating multiple *Actions*.

(a) Action setting

(b) Object setting

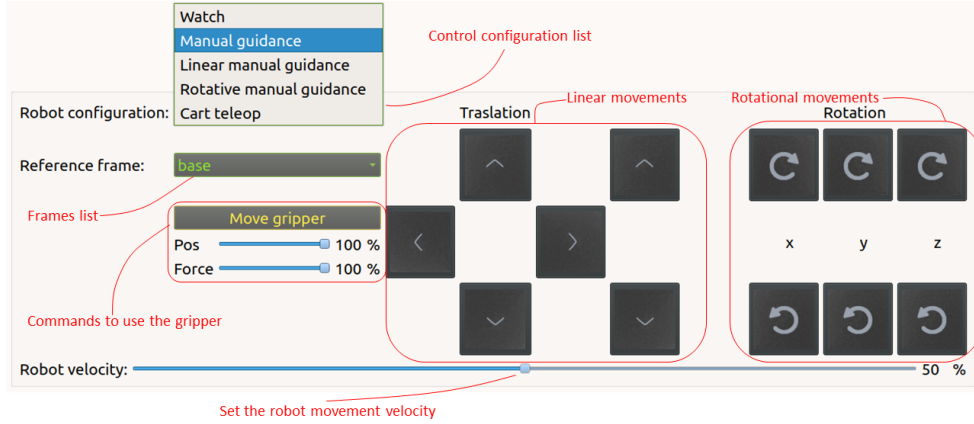Figure 6. Action and components setting panels.

Figure 7. Interface Joystick, the movements are performed moving the end-effector frame with respect to the chosen reference frame.

in case of unfeasible inverse kinematics or motion plan, so it is enabled the modification of the components or the *Action* sequence to fix the issue. Currently, the framework provides only three actions. The development of a new one still requires an expert developer; however, the number of actions needed to address complex industrial tasks is limited (Bøgh et al., 2012), and the actions available in the MF allow it to cover a wide range of tasks. The possibility to intuitively generate a custom action, starting with the elementary skills, will be part of future works.

3.2. Experimental Setup

The experimental setup consists of a Universal Robots UR10e mounted on a Cobotracks linear guide. The robot gripper is a Robotiq 2F-85. An external PC controls the robot at a frequency of $500[Hz]$. The software used for the PC robot communication is the official ROS package (Universal Robots, 2022). This package requires the micro-interpolated joint positions to control the robot. The feedback information are joint positions, velocities, currents and external forces.

The robot's end-effector is a Robotiq gripper with two fingers and $85[mm]$ of working range; a force/torque sensor was mounted between the robot flange and the gripper. The force measuring range is $100[N]$, and the torque measuring range is $10[Nm]$. Figure 8 shows the experimental setup, highlighting the ceil robot configuration. The Cobotracks linear guide was not used during the experiments.

3.3. Experiments description

During the experimental session, the users programmed two tasks exploiting the *robot-oriented* (*i.e.,* the UR10 Teach Pendant Interface, TPI hereafter) and the *task-oriented* (*i.e.,* the MFI) programming interfaces. Task 1 was a pick&place task with 10 objects, thought to measure the performance (*i.e.,* programming and testing time) of the two programming interfaces with a highly repetitive task. Task 2 recalls a machine tending application where the robot has to enter the machine tool workspace and withdraw or release a workpiece by avoiding obstacles. The task was thought to measure the performance of the interfaces when required to deal with multiple obstacles in the robot workspace, restricted spaces, and reprogramming.
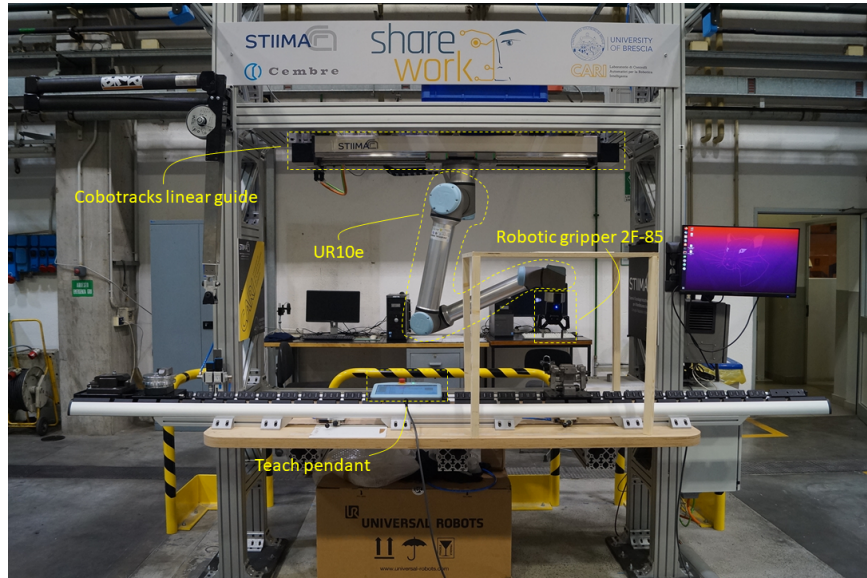
10

Figure 8. Experiment setup.

The experiments consist of five phases:

(1) introduction: the user is informed about the experiment and the test phases.
(2) Teaching: the user watches a video that describes the interfaces (*i.e.,* the robot TPI or the MFI) and their usage. Then, an expert operator supports the user in assisted training, where the goal is to perform a single pick and place. In this phase, the user is free to ask questions to the trainer. The training continues until the user declares he/she can program a task autonomously. Finally, the expert operator describes the user's task to program; the programming phase can start.
(3) Autonomous programming: the user programs the robot without the help of an expert. This phase ends when the user declares finished the task programming. The user can ask questions if he/she cannot proceed in task programming. The number of questions is an evaluation parameter.
(4) Testing: testing the program developed in the previous phase. In case the task is correctly performed, this phase ends. On the contrary, the user must correct and test the program until it is completed. The task's success determines the end of this phase.
(5) Questionnaire: the user fills in a questionnaire regarding the intuitiveness and complexity of the interface.

The experiments on Tasks 1 and 2 were carried out with 22 people. All subjects participated voluntarily, signing an informed consent form by the Declaration of Helsinki.

3.3.1. Task 1 experiment description

Task1 requires the pick&place of ten objects. The objects were of two types (cubes and cylinders), 5 objects per type. After the picking, the robot place the object in the relative box on the base of the object type. The programmer is free to decide the picking sequence and, if necessary, trajectories via-points. Figure 9 shows the objects and boxes layout. The box on the right was for the cubes, and the box on the left was
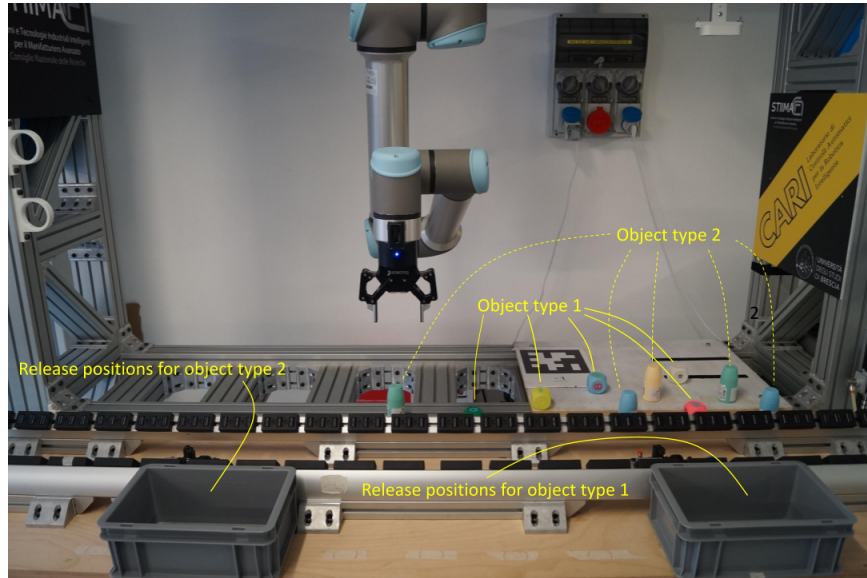
11

Figure 9. *Task 1 setup and description.*

for the cylinders.

The experiments enlisted two groups of users, hereafter named Group A and Group B. Users were mostly university students from STEM faculties, aged between 20 and 35 years old. Group A was composed of 8 people; they programmed the application using the UR10e TP's described in Section 3.1.1. Group B was composed of 9 people; they programmed the same application using the MFI described in Section 3.1.2.

The experiment consists of the five phases described in Section 3.3.

3.3.2. Task 2 experiment description

Task 2 requires picking 2 objects from a constrained box, moving them outside the box and placing them in a specific area. The picking sequence is free; the programmer can decide the picking positions. Once the program is finished and tested, the programmer has to change the release positions of the objects.

The idea is to avoid a penalisation of the TPI when a high number of repetitive *Actions* is required, as for Task 1. Indeed, the task involves only 2 objects. At the same time, the robot has to move through a restricted environment with obstacles to evaluate the planning performance of the MF (*i.e.,* collision-free motion planning) compared to the TP where the programmer has to take care of the collision avoidance by adding intermediate trajectory via-points.

This task simulates a typical application of an industrial robot in machine tending, where the robot has to enter the machine tool workspace to withdraw or release a workpiece, and several obstacles constrain the planning environment. The programmer has to take care of possible collisions carefully. A frequent request is the reprogramming of robot tasks due to modifications to the robotic cell. Indeed, at the end of the task programming, we requested to adjust the robot program slightly to deal with robotic cell changes. We asked to partially modify the original robot program to measure the reprogramming time with both interfaces. These requests are frequent when small batches and huge production variability characterise the production.

The experiments' execution was on the shop floor of a mechanical engineering com-
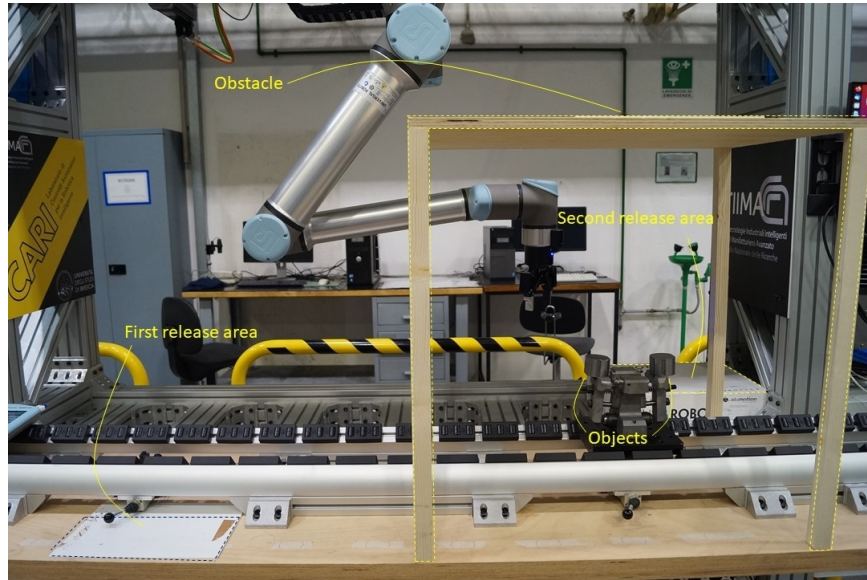
Figure 10. *Task 2 setup and description.*

pany. The experiment participants were machine tools operators and one production engineer with neither a robotic background nor previous experience in robot programming; it was the first time they had seen both programming interfaces. The users' group was composed of 5 people, aged between 26 and 46 years old; 4 over 5 users have high school graduation, while the fifth has an automation engineering degree. The users were all with technical backgrounds and experiences in machine tool programming but no experience in industrial robot use.

The hosting company provided only a limited number of users, so it was impossible to create two independent groups; thus, all the participants repeated the experiments twice using both interfaces. The first experiment can influence the learning phase of the second one because the programming experience can provide some knowledge of the task for the second experiment; the order of the interfaces was randomly alternated, avoiding any possible bias. The users who experimented with Task 2 were not involved in Task 1 experiments' and vice-versa.

This experiment was composed of the five phases described in Section 3.3, with the addition of an autonomous reprogramming phase after phase 4 where the user has to modify the program already developed. The time spent by the user during the autonomous reprogramming and the number of questions were recorded as evaluation parameters. A second test phase has been added to correct any reprogramming errors. Finally, the user has to fill the questionnaire.

3.4. *Performance Measurement*

In general, during the task programming, it is possible to identify 4 main phases: (i) the user learning phase, (ii) the program development phase, (iii) the testing phase, (iv) the execution phase. The first evaluation metric is the time taken by these phases. Therefore, the following evaluation parameters have been chosen:

- learning time (LeT): the time spent by the user to learn the programming environment during the assisted programming in phase 2 (teaching). LeT evaluates

Table 1. Questionnaire proposed to users after Task 1 and Task 2.

1	On a scale of 1 to 10, where 1 is inexperienced and 10 is very experienced, how experienced are you in the use of industrial robots?
2	On a scale of 1 to 10, where 1 is not simple and 10 is very simple, how simple did you find the use of the programming interface you were proposed to use?
3	On a scale of 1 to 10, where 1 is not intuitive and 10 is very intuitive, how intuitive did you find the use of the programming interface you were proposed to use?
4	On a scale of 1 to 10, where 1 is not fast and 10 is very fast, how fast did you find learning to use the programming interface you were proposed to use?
5	On a scale of 1 to 10, where 1 is not useful 10 is very useful, how much did your previous knowledge affect you in learning to use the programming interface you were proposed to use?

the steepness of the learning curve of the interface.

- Programming time ($\boldsymbol{PrT}$): the time spent by the user to develop a robot program to achieve the given task. PrT reflects the ease of use and the intuitiveness of the interface.
- Testing time ($\boldsymbol{TeT}$): the time spent by the user to test the robot program. The programmer must test the program developed in the real working environment by checking possible collisions and adding or adjusting via-points if needed. TeT is a proxy of the ability of the interface to avoid user errors during task programming as it measures the time spent for program debugging.
- Execution time ($\boldsymbol{ExT}$): the time spent by the robot to execute the program. The execution time is related to the robot speed (equal for both the interfaces) and the length of the paths found by the motion planners. Using the TP, the ExT depends on the number and the position of the via-points selected by the programmer; using the MFI the length of the trajectory is defined by the selected motion planner.
- Number of tests executed ($\boldsymbol{TeN}$): the number of executions run before a correct task execution (excluded ExT). Similarly to TeT, TeN evaluates the robustness of the programming interface w.r.t. the errors made by the user during the programming.
- Number of questions during programming ($\boldsymbol{PrQ}$) and testing ($\boldsymbol{TeQ}$): the amount of questions highlights the user understanding of the interface use, it evaluate the ease of use and the intuitiveness of the interface;

During the experiments of *Task 2*, the following parameters were also measured to evaluate the reprogramming phase:

- reprogramming time ($\boldsymbol{ReT}$): the same as PrT during the re-programming phase.
- reprogramming testing time ($\boldsymbol{ReTeT}$): the same as TeT during the re-programming phase.
- reprogramming execution time ($\boldsymbol{ReExT}$): the same as ExT during the re-programming phase.
- number of tests made during the reprogramming ($\boldsymbol{ReTeN}$): the same as TeN during the re-programming phase.
- number of questions during reprogramming ($\boldsymbol{ReQ}$) and reprogramming testing ($\boldsymbol{ReTeQ}$);

Finally, to evaluate the user feel, users filled out the questionnaire in Table 1.
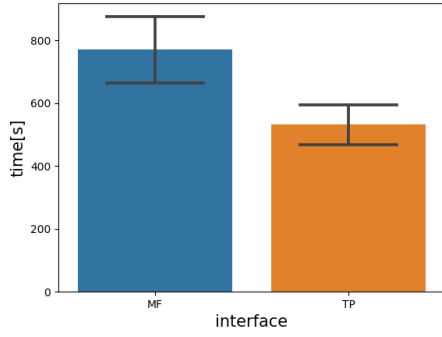
4. Results and Discussion

The methodology applied during experiment execution is described in Section 3.3. The evaluation exploits the performance indexes described in Section 3.4.
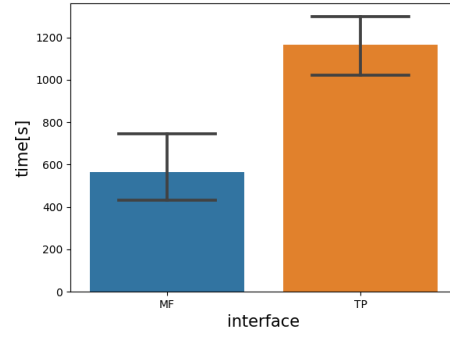
4.1. *Task 1 Experiments Results*

The experiments relative to Task 1 applied the method described in Section 3.3 and 3.3.1. Section 3.4 reports the parameters measured to evaluate the interfaces. All the evaluation indices measured during Task 1 are reported in Figures 11 and 12.

The average learning time LeT for the MFI is 44.9% higher than the TPI, see Figure 11a. This result was expected as the MFI has more complex concepts than the TPI and the learning time tends to be higher. The average programming time PrT for the MFI is 51.7% lower than the TPI, see Figure 11b. High values of PrT for the TPI are directly related to the number of objects involved in the task because the operator needs to teach multiple positions to perform collision-free trajectories. On the contrary, the MFI allows defining a *Pick&Place Task* teaching only two grasping positions to define the *Pick Action* and two release positions to define the *Place Action*. Once defined as an *Action*, the operator can replicate the same *Action*. The average number of programming questions $PrQs$ and test questions *TeQs*, see Figures 11c and 11f respectively, is low for both the interfaces. These low values represent a good operator learning rate that reflects comparable ease of use for both interfaces. The number of tests *TeNs* and the test time TeT, see Figure 11e and 11d respectively, are low. Most of the experiments do not present mistakes during the programming. The low presence of mistakes avoids corrections in many experiments; when required, the correction time is short, strengthening the result already given by PrT values. The low number of mistakes leads to overlapping the *TeTs* and the *ExTs*. The average execution time ExT is 13.5% lower for the TPI, see Figure 11g. The robot motion planner interpolates the trajectory via-points taught through the TPI. The time to compute the trajectories is short, and the via-points are linearly interpolated. Instead, the MFI interpolates the starting and the goal position with optimal collision-free trajectories. The computational time to evaluate the planning scene and generates the collision-free trajectories can vary. This difference explains the differences in the execution times. Despite this, the difference between the results is insignificant; furthermore, TPI ExT present a larger standard deviation than the MFI. The large standard deviation of the ExT of TPI highlights a high dependency on the user's skills. The MFI presents a small ExT standard deviation because the execution is independent of the operator's capacity.
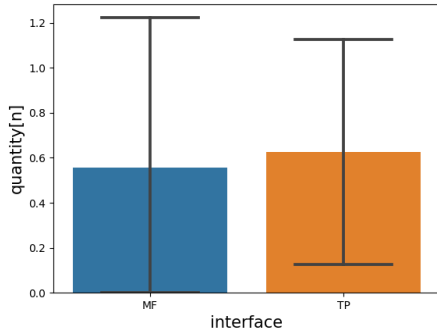
In the end, a questionnaire was proposed to the users (see Table 1). The questions are related to the user's impressions concerning his/her previous experience level (Q1), ease of use (Q2), intuitiveness (Q3), learning speed (Q4), and the utility of his/her previous knowledge (Q5). Figure 11h, 12a and 12b show almost identical results between the MFI and the TPI regarding ease of use, intuitiveness and learning speed. The plots of the score given to questions 1 and 5 (user's previous experience level and the utility of the previous knowledge to program the task) are not reported because the scores were always low. Therefore the informative contribution was not significant. Table 2a, 2b shows information about the users that participated in the experiments of Task 1.
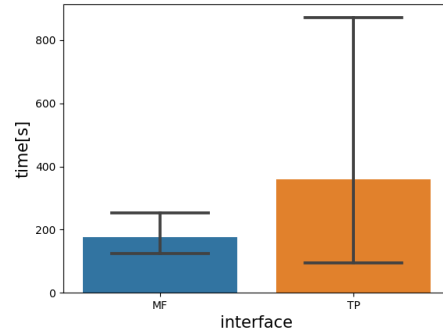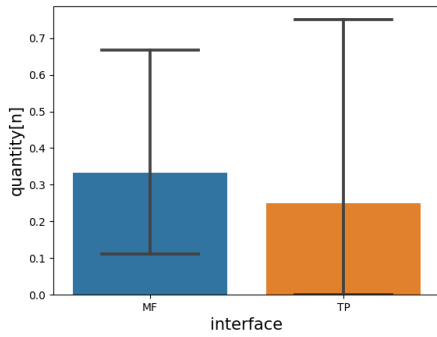
(a) *Learning Time (LeT).*
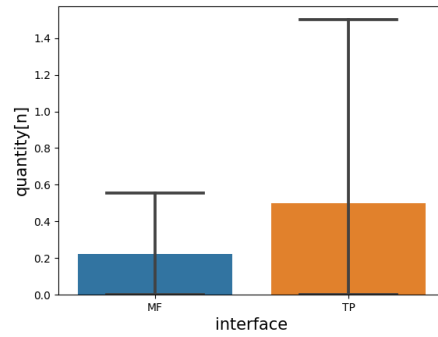
(b) *Programming Time (PrT).*
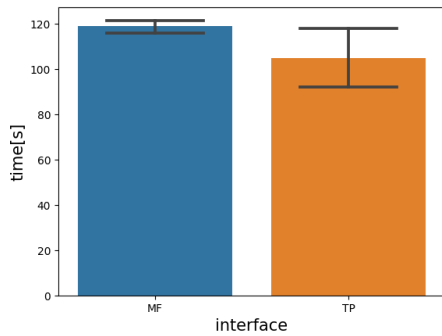
(c) *Programming Questions (PrQs).*
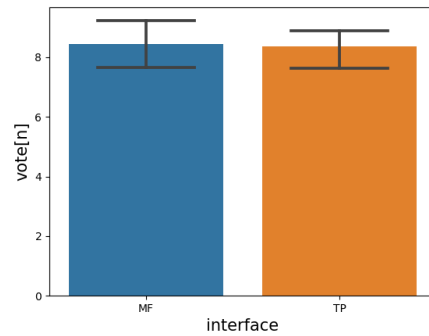
(d) *Test Time (TeT).*

(e) *Test Numbers (TeNs).*
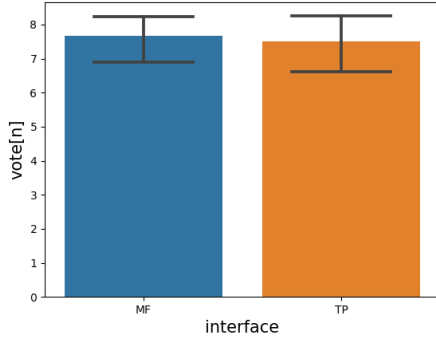
(f) *Test Questions (TeQs).*
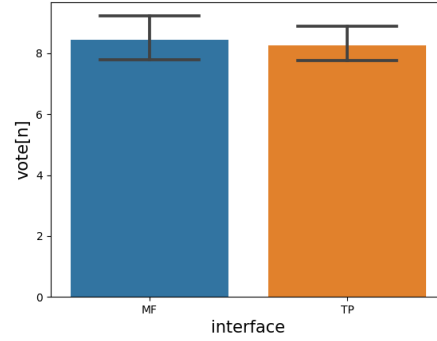
(g) *Execution Time (ExT).*

(h) *Interface ease of use, Table 1 Q2.*

Figure 11. *Task 1* experiments' results.
TP: UR10e teach pendant interface. MF: manipulation framework interface.

(a) *Interface intuitiveness, Table 1 Q3.*   (b) *Interface speed, Table 1 Q4.*

Figure 12. *Task 1* experiments' results.
TP: UR10e teach pendant interface. MF: manipulation framework interface interface.

Table 2. Participants data for Task 1: E.Q. (educational qualification): h.s.g. (high school graduation); m.d. (master's degree); b.d. (bachelor degree). I.R.P.E. (industrial robot programming experience). P.O. (professional occupation).

(a) Group A

	Age	E.Q.	I.R.P.E.	P.O.	Gender
1	24	m.d.	little	PhD student	man
2	28	m.d.	no	PhD student	man
3	35	h.s.g.	little	mechanical operator	man
4	23	b.d.	no	student	man
5	20	h.s.g.	no	student	female
6	23	m.d.	little	student	man
7	26	m.d.	no	PhD student	man
8	22	h.s.g.	no	student	man

(b) Group B

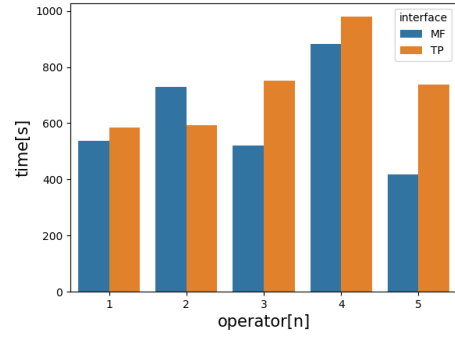|	Age	E.Q.	I.R.P.E.	P.O.	Gender
9	29	PhD	expert	researcher	male
10	35	h.s.g.	little	mechanical operator	male
11	23	b.d.	no	student	female
12	24	b.d.	little	student	male
13	24	b.d.	little	student	male
14	23	b.d.	no	student	female
15	22	h.s.g.	no	student	male
16	31	m.d.	no	sw developer	male
17	34	m.d.	no	sw engineer	male

4.2. *Task 2 Experiments Results*

The experiments relative to *Task 2* applied the method described in Section 3.3 and 3.3.2. Section 3.4 reports the parameters measured to evaluate the interfaces. All the evaluation indices measured during Task 2 are reported in Figures 13 and 14.

The average learning time LeT is 43.7% higher for MFI; see Figure 13a. The result is similar to the LeT of Task 1 experiments. The average programming time PrT is 15.2% lower for the MFI; see Figure 13b. The PrT is similar for both the interfaces because *Task 2* is composed of only two objects, and the number of repetitive *Actions* is reduced, tending to provide similar results. The programming questions $PrQs$ is higher for the MFI; see Figure 13c. The users involved in Task 2 (*i.e.,* shop floor machine tools operators and technicians) had less familiarity with the use of robots and, in general, less inclination to technologies compared to the users of *Task 1* (*i.e.,* STEM faculties students). This aspect is the possible cause of why *Task 2 PrQs* values are higher than *Task 1*, in particular for the MFI. The more complex structure of MFI has amplified this phenomenon. Despite this problem, the $PrQs$ values do not represent a real problem. The training of non-expert operators for the MFI is less than one hour (considering the video watching). The test time TeT, see Figure 13d, shows similar TeT for both interfaces apart from the spike of user one that is however, present for both interfaces; in particular, the time to correct the errors is comparable. The test numbers $TeNs$, see Figure 13e, show a not relevant number of trials, so the reduced number of errors made during the programming demonstrates that reduced knowledge does not affect the operator performance. The test questions $TeQs$, see Figure 13f, show a high autonomy of the user to correct the errors. The execution time ExT, see Figure 13g, indicates that the TPI ExT has lower values than the MFI. In this case, the computation time necessary for the MF to generate collision-free trajectories is higher than Task 1 because the robot workspace presents constrained spaces and more obstacles. The higher $ExTs$ is reflected in the benefit of a collision-free trajectory guaranteed by the MF motion planners. On the contrary, with the TPI, the collision avoidance of the robot is in charge of the programmer. The average reprogramming time ReT, see Figure 13h, shows that usually, the MFI ReT values are lower than the TPI. The MFI ReT average value is 26.1% lower than TPI. With the MFI, the user has to modify the program to teach only two new positions. Instead, the TPI requires adding new via-points to the trajectories already defined. The reprogramming number of questions $ReQs$, see Figure 14a, shows the low $ReQs$ values; most of the users did not need any help during the reprogramming. The reprogramming test times $ReTeTs$, the reprogramming tests numbers $ReTeN$, the test questions $ReTeQs$, and the execution time $ReExT$, see Figure 14b, 14c, 14d, and 14e respectively, show results similar to those obtained in the first testing phase and no significant differences emerged between the interfaces.
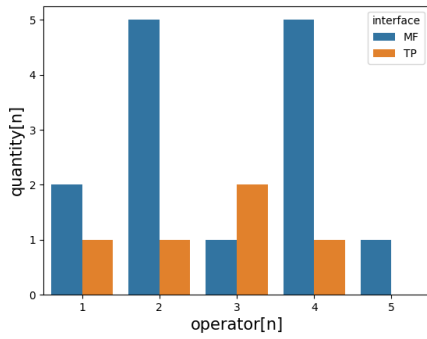
As for Task 1, at the end of Task 2, the questionnaire described in Table 1 was proposed to the users. Figure 14f, 14g, and 14h show again similar values for both interfaces. The parameters have high values demonstrating a good appreciation by the machine tools operators highlighting the usability of the interfaces in the industrial world. The assessments of previous knowledge have no informative contribution, and the plot of the scores of questions 1 and 5 are not reported. Table 3 show the information about the users participating in the experiments of Task 2.
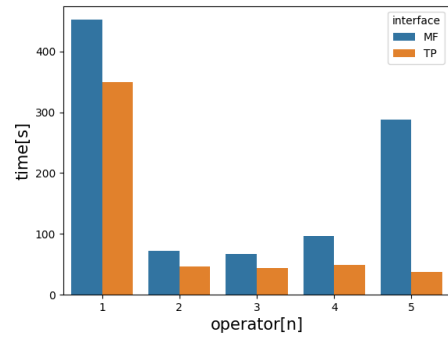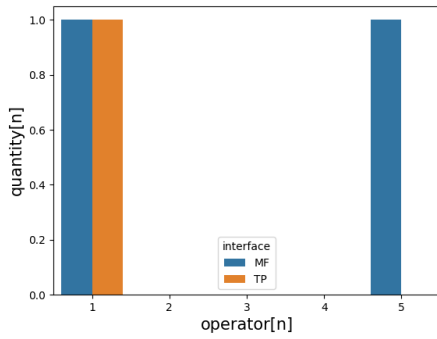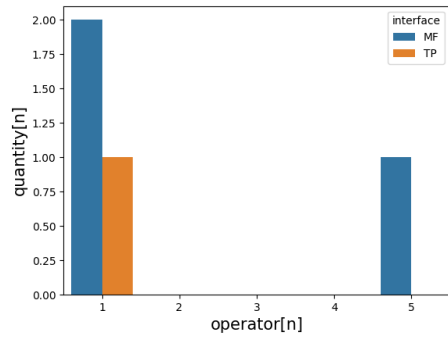
(a) *Learning Time (LeT).*

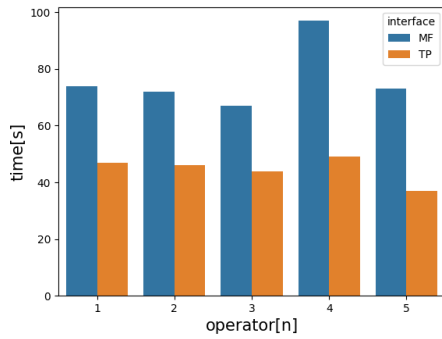(b) *Programming Time (PrT).*

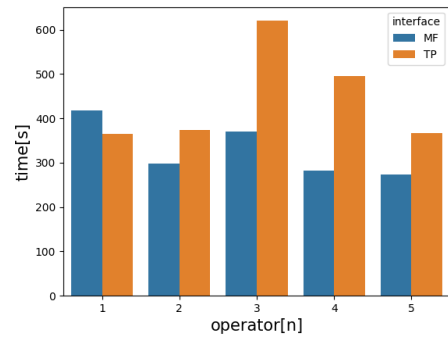(c) *Programming Questions (PrQs).*

(d) *Test Time (TeT).*

(e) *Test Numbers (TeNs).*
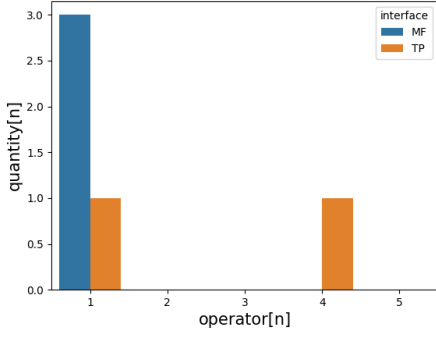
(f) *Test Questions (TeQs).*
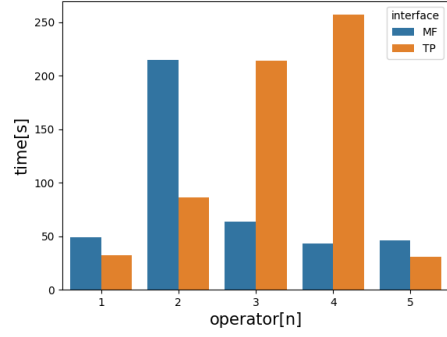
(g) *Execution Time (ExT).*

(h) *Reprogramming Time (ReT).*

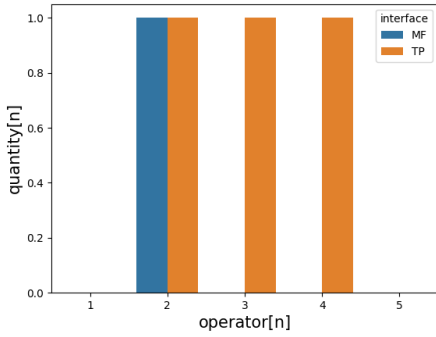Figure 13. *Task 2* experiments' results.
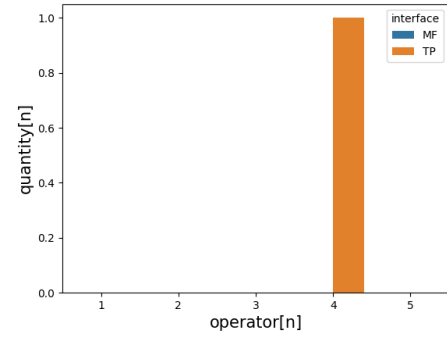TP: UR10e teach pendant. MF: manipulation framework interface.

(a) *Reprogramming Questions (ReQs).*

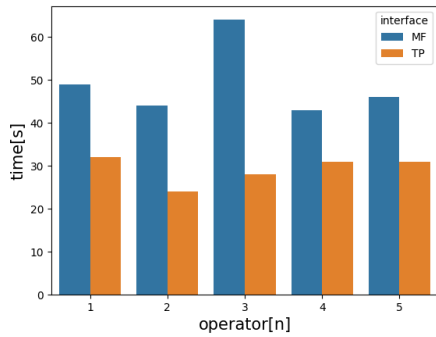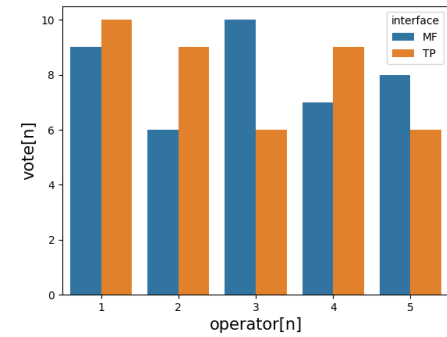(b) *Reprogramming Test Time (ReTeT).*
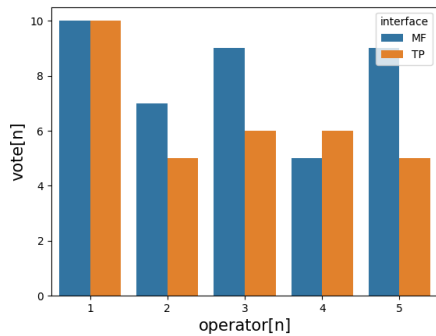
(c) *Reprogramming Test Numbers (ReTeN).*

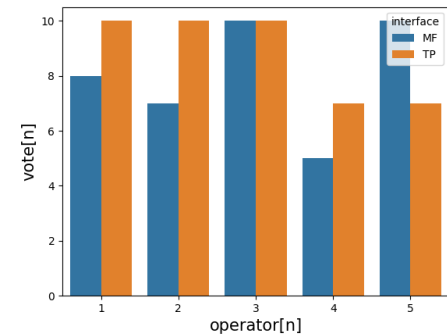(d) *Reprogramming Test Questions (ReTeQ).*

(e) *Reprogramming Execution Time (ReExT).*

(f) *Interface ease of use, Table 1 Q2.*

(g) *Interface intuitiveness, Table 1 Q3.*

(h) *Interface speed, Table 1 Q4.*

Figure 14. *Task 2* experiments' results.
TP: UR10e teach pendant. MF: manipulation framework interface.

Table 3. Participants data for Task 2: E.Q. (educational qualification): h.s.g. (high school graduation); m.d. (master's degree); b.d. (bachelor degree). I.R.P.E. (industrial robot programming experience). P.O. (professional occupation).

	Age	E.Q.	I.R.P.E.	P.O.	Gender
1	36	h.s.g.	little	mechanical operator	male
2	35	h.s.g.	little	mechanical operator	male
3	32	h.s.g.	No	mechanical operator	male
4	46	h.s.g.	No	shift supervisor	male
5	26	m.d.	little	process engineer	male

4.3. *Discussion*

The experiments presented similar results for both interfaces. The similarities are visible in every operational aspect of the interfaces. In other words, the MFI can compete with the UR10e TPI. The higher learning times of the MFI can be quickly recovered by the daily use of the MF for robot programming when the tasks present repetitive *Actions*. However, the learning time of the MFI for a non-expert end-user was minimal and below one hour to give a basic understanding of the interface and to be independent of the task programming. At the same time, the MF brings together several additional benefits, such as using the ROS framework, *task-oriented* programming, and collision-free motion planning. The goal of using a complex robot programming framework accompanied by an intuitive interface was reached, demonstrating that shop floor operators can quickly adopt the framework without significant learning barriers and any particular preliminary knowledge. This paper does not present an alternative to UR10e TP. The paper aims to demonstrate that using complex software with a suitable programming interface can bring advanced programming features even to shop floor operators, enabling the use of advanced robotic cells in SMEs.

5. Conclusions and Future Works

This paper compares two robot programming interfaces representing two different robot programming approaches. The first is the UR10e teach pendant interface, combined with *lead-through* programming, enabling a classic *robot-oriented* approach. The second is the Manipulation Framework interface providing *task-oriented* programming approach. The experiments over 22 users show similar results for both interfaces highlighting that an intuitive interface that hides the complexity of a framework based on ROS, such as the MF, can reach a high level of acceptance between end-users without specific programming experience. The short learning times show the possibility of training an end-user in very little time bringing advanced features to the shop floor without particular knowledge of robotics. At the same time, the flexibility during reprogramming improved. The modification of the robot control program can be accessible to all production plant operators.

In future work, the authors will investigate using machine learning algorithms to create new actions. From a limited number of skills, for example, machine learning algorithms should dynamically compose new actions based on specific requirements without developer interventions.

Acknowledgement

References

Ajaykumar, Gopika, Maureen Steele, and Chien-Ming Huang. 2021. "A Survey on End-User Robot Programming." *ACM Comput. Surv.* 54 (8). `https://doi.org/10.1145/3466819`.

Ajaykumar, Gopika, Maia Stiber, and Chien-Ming Huang. 2021. "Designing user-centric programming aids for kinesthetic teaching of collaborative robots." *Robotics and Autonomous Systems* 145: 103845. `https://www.sciencedirect.com/science/article/pii/S0921889021001305`.

Andronas, Dionisis, George Apostolopoulos, Nikos Fourtakas, and Sotiris Makris. 2021. "Multi-modal interfaces for natural Human-Robot Interaction." *Procedia Manufacturing* 54: 197–202. 10th CIRP Sponsored Conference on Digital Enterprise Technologies (DET 2020) – Digital Technologies as Enablers of Industrial Competitiveness and Sustainability, `https://www.sciencedirect.com/science/article/pii/S2351978921001669`.

Anzalone, Salvatore M., Marie Avril, Hanan Salam, and Mohamed Chetouani. 2014. "IMI2S: A Lightweight Framework for Distributed Computing." In *Simulation, Modeling, and Programming for Autonomous Robots*, edited by Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, Cham, 267–278. Springer International Publishing.

Apostolopoulos, George, Dionisis Andronas, Nikos Fourtakas, and Sotiris Makris. 2022. "Operator training framework for hybrid environments: An Augmented Reality module using machine learning object recognition." *Procedia CIRP* 106: 102–107. 9th CIRP Conference on Assembly Technology and Systems, `https://www.sciencedirect.com/science/article/pii/S2212827122001639`.

Bascetta, Luca, Gianni Ferretti, Gianantonio Magnani, and Paolo Rocco. 2013. "Walk-through programming for robotic manipulators based on admittance control." *Robotica* 31 (7): 1143–1153.

Billard, A., S. Calinon, R. Dillmann, and S. Schaal. 2008. "Robot Programming by Demonstration. In: Handbook of Robotics." .

Blankemeyer, Sebastian, Rolf Wiemann, Lukas Posniak, Christoph Pregizer, and Annika Raatz. 2018. "Intuitive Robot Programming Using Augmented Reality." *Procedia CIRP* 76: 155–160. 7th CIRP Conference on Assembly Technologies and Systems (CATS 2018), `https://www.sciencedirect.com/science/article/pii/S2212827118300933`.

Bøgh, Simon, Oluf Skov Nielsen, Mikkel Rath Pedersen, Volker Krüger, and Ole Madsen. 2012. "Does your robot have skills?" In *Proceedings of the 43rd international symposium on robotics*, VDE Verlag GMBH.

Castro, André, João Pedro Souza, Luís Rocha, and Manuel F. Silva. 2019. "Adapt-Pack Studio: Automatic Offline Robot Programming Framework for Factory Envi-

ronments." In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 1–6.

Cembre S.p.A. 2022. "Cembre S.p.A." `https://www.cembre.it/`.

Coronado, Enrique, Fulvio Mastrogiovanni, Bipin Indurkhya, and Gentiane Venture. 2020. "Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review." *Journal of Computer Languages* 58: 100970. `https://www.sciencedirect.com/science/article/pii/S2590118420300307`.

Delledonne, Michele, and Manuel Beschi. 2022a. "Manipulation interface." `https://github.com/JRL-CARI-CNR-UNIBS/manipulation_interface`.

Delledonne, Michele, and Manuel Beschi. 2022b. "Manipulation interface." `https://github.com/JRL-CARI-CNR-UNIBS/manipulation_interface_example`.

Faroni, Marco, Manuel Beschi, Stefano Ghidini, Nicola Pedrocchi, Alessandro Umbrico, Andrea Orlandini, and Amedeo Cesta. 2020. "A Layered Control Approach to Human-Aware Task and Motion Planning for Human-Robot Collaboration." In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 1204–1210.

Ferraguti, Federica, Chiara Talignani Landi, Lorenzo Sabattini, Marcello Bonfè, Cesare Fantuzzi, and Cristian Secchi. 2019. "A variable admittance control strategy for stable physical human–robot interaction." *The International Journal of Robotics Research* 38 (6): 747–765. `https://doi.org/10.1177/0278364919840415`.

Gadre, Samir Yitzhak, Eric Rosen, Gary Chien, Elizabeth Phillips, Stefanie Tellex, and George Konidaris. 2019. "End-User Robot Programming Using Mixed Reality." In *2019 International Conference on Robotics and Automation (ICRA)*, 2707–2713.

Heimann, Oliver, and Jan Guhl. 2020. "Industrial Robot Programming Methods: A Scoping Review." In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1, 696–703.

Huang, Gaoping, Pawan Rao, Meng-Han Wu, Xun Qian, Shimon Nof, Karthik Ramani, and Alexander Quinn. 2020. "Vipo: Spatial-Visual Programming with Functions for Robot-IoT Workflows." In *CHI '20: CHI Conference on Human Factors in Computing Systems*, 04, 1–13.

Huang, Justin, and Maya Cakmak. 2017. "Code3: A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts." In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, 453–462.

International Federation of Robotics. 2022. "World Robotics Industrial Robots Report." `https://ifr.org/worldrobotics/`.

ISO/TC 299 Robotics. 2016. "ISO/TS 15066:2016." `https://www.iso.org/standard/62996.html`.

Kubota, Alyssa, Emma I. C. Peterson, Vaishali Rajendren, Hadas Kress-Gazit, and Laurel D. Riek. 2020. *JESSIE: Synthesizing Social Robot Behaviors for Personalized Neurorehabilitation and Beyond*, 121–130. New York, NY, USA: Association for Computing Machinery. `https://doi.org/10.1145/3319502.3374836`.

Kuka AG. 2022. "Kuak Sunrise.OS." `https://www.kuka.com/en-gb/products/robotics-systems/software/system-software/sunriseos`.

Lu, Shuang, Julia Berger, and Johannes Schilp. 2022. "System of Robot Learning from Multi-Modal Demonstration and Natural Language Instruction." *Procedia CIRP* 107: 914–919. Leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022, `https://www.sciencedirect.com/science/article/pii/S2212827122003687`.

Makris, Sotiris, Panagiota Tsarouchi, Dragoljub Surdilovic, and Jörg Krüger. 2014. "Intuitive dual arm robot programming for assembly operations." *CIRP An-*

nals 63 (1): 13–16. `https://www.sciencedirect.com/science/article/pii/S0007850614000201`.

Metta, Giorgio, Paul Fitzpatrick, and Lorenzo Natale. 2006. "YARP: Yet Another Robot Platform." *International Journal of Advanced Robotic Systems* 3 (1): 8. `https://doi.org/10.5772/5761`.

Michalos, George, Sotiris Makris, Jason Spiliotopoulos, Ioannis Misios, Panagiota Tsarouchi, and George Chryssolouris. 2014. "ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future." *Procedia CIRP* 23: 71–76. 5th CATS 2014 - CIRP Conference on Assembly Technologies and Systems, `https://www.sciencedirect.com/science/article/pii/S2212827114011366`.

Mohammad Safeea, Pedro Neto. 2022. "Precise positioning of collaborative robotic manipulators using hand-guiding." *The International Journal of Advanced Manufacturing Technology* 120: 5497–5508. `https://doi.org/10.1007/s00170-022-09107-1`.

Mukherjee, Debasmita, Kashish Gupta, Li Hsin Chang, and Homayoun Najjaran. 2022. "A Survey of Robot Learning Strategies for Human-Robot Collaboration in Industrial Settings." *Robotics and Computer-Integrated Manufacturing* 73: 102231. `https://www.sciencedirect.com/science/article/pii/S0736584521001137`.

Neto, Pedro, and Nuno Mendes. 2013. "Direct off-line robot programming via a common CAD package." *Robotics and Autonomous Systems* 61 (8): 896–910. `https://www.sciencedirect.com/science/article/pii/S0921889013000419`.

Ong, S.K., A.W.W. Yew, N.K. Thanigaivel, and A.Y.C. Nee. 2020. "Augmented reality-assisted robot programming system for industrial applications." *Robotics and Computer-Integrated Manufacturing* 61: 101820. `https://www.sciencedirect.com/science/article/pii/S0736584519300250`.

Open source community. 2022. "MoveIt! Moving robots into the future." `https://moveit.ros.org/`.

Pan, Zengxi, Joseph Polden, Nathan Larkin, Stephen Van Duin, and John Norrish. 2012. "Recent progress on programming methods for industrial robots." *Robotics and Computer-Integrated Manufacturing* 28 (2): 87–94. `https://www.sciencedirect.com/science/article/pii/S0736584511001001`.

Quintero, Camilo Perez, Sarah Li, Matthew KXJ Pan, Wesley P. Chan, H.F. Machiel Van der Loos, and Elizabeth Croft. 2018. "Robot Programming Through Augmented Trajectories in Augmented Reality." In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1838–1844.

Rendiniello, Angelo, Alberto Remus, Ines Sorrentino, Prajval Kumar Murali, Daniele Pucci, Marco Maggiali, Lorenzo Natale, et al. 2020. "A Flexible Software Architecture for Robotic Industrial Applications." In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vol. 1, 1273–1276.

ROBO-PARTNER Consortium. Nov. 2013 - Apr. 2017. "ROBO-PARTNER Project Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future." `http://www.robo-partner.eu/`.

ROS-Industrial consortium. 2022. "ROS-Industrial." `https://rosindustrial.org/`.

Schmidt, Douglas C., David L. Levine, and Sumedh Mungee. 1998. "The Design of the TAO Real-Time Object Request Broker." *Comput. Commun.* 21 (4): 294–324. `https://doi.org/10.1016/S0140-3664(97)00165-5`.

Schou, Casper, Rasmus Skovgaard Andersen, Dimitrios Chrysostomou, Simon Bøgh, and Ole Madsen. 2018. "Skill-based instruction of collaborative robots in industrial settings." *Robotics and Computer-Integrated Manufacturing* 53: 72–80. `https://www.sciencedirect.com/science/article/pii/S0736584516301910`.

Sefidgar, Yasaman S., Prerna Agarwal, and Maya Cakmak. 2017. "Situated Tangible Robot Programming." In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, New York, NY, USA, 473–482. Association for Computing Machinery. `https://doi.org/10.1145/2909824.3020240`.

Sharework Consortium. Nov. 2018 - Oct. 2022. "Sharework." `https://sharework-project.eu/`.

Sherlock Consortium. Oct. 2018 - Sep. 2022a. "Sharlock - Seamless and safe human centred robotic applications for novel collaborative workplaces." `https://www.sherlock-project.eu/home`.

Sherlock Consortium. Oct. 2018 - Sep. 2022b. "Sherlock - Deliverable 4.8 Autonomous learning and programming methods – Final prototype." `https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5e30cb5db&appId=PPGMS`.

Siciliano, Bruno, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. 2010. "Robotics: modelling, planning and control." .

Stanford Artificial Intelligence Laboratory et al. 2022. "Robotic Operating System." `https://www.ros.org`.

Stenmark, Maj, Mathias Haaae, and Elin Anna Topp. 2017. "Simplified Programming of Re-Usable Skills on a Safe Industrial Robot - Prototype and Evaluation." In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, 463–472.

Thormann, Christian, and Alexander Winkler. 2021. "Programming of a Lightweight Robot Using Function Blocks and Sequential Function Charts." In *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*, 348–353.

Tsarouchi, Panagiota, Sotiris Makris, and George Chryssolouris. 2016. "Human–robot interaction review and challenges on task planning and programming." *International Journal of Computer Integrated Manufacturing* 29 (8): 916–931. `https://doi.org/10.1080/0951192X.2015.1130251`.

Umbrico, Alessandro, Andrea Orlandini, Amedeo Cesta, Marco Faroni, Manuel Beschi, Nicola Pedrocchi, Andrea Scala, et al. 2022. "Design of Advanced Human–Robot Collaborative Cells for Personalized Human–Robot Collaborations." *Applied Sciences* 12 (14): 6839. `http://dx.doi.org/10.3390/app12146839`.

Universal Robots. 2022. "Universal Robots ROS Driver." `https://github.com/UniversalRobots/Universal_Robots_ROS_Driver`.

Villagrossi, E., N. Pedrocchi, and M. Beschi. 2021. "Simplify the robot programming through an action-and-skill manipulation framework." In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–6.

Villagrossi, Enrico, and Manuel Beschi. 2022a. "Manipulation examples." `https://github.com/JRL-CARI-CNR-UNIBS/manipulation_examples`.

Villagrossi, Enrico, and Manuel Beschi. 2022b. "The manipulation library." `https://github.com/JRL-CARI-CNR-UNIBS/manipulation`.

Villani, Valeria, Fabio Pini, Francesco Leali, and Cristian Secchi. 2018. "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications." *Mechatronics* 55: 248–266. `https://www.sciencedirect.com/science/article/pii/S0957415818300321`.

Yang, Shuo, Xinjun Mao, Binbin Ge, and Sen Yang. 2015. "The Roadmap and Challenges of Robot Programming Languages." In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 328–333.

Zhang, Jiafan, Yue Wang, and Rong Xiong. 2016. "Industrial robot programming by demonstration." In *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, 300–305.