Robotics and Computer–Integrated Manufacturing 89 (2024) 102770

Contents lists available at ScienceDirect

Robotics and Computer-Integrated Manufacturing

journal homepage: www.elsevier.com/locate/rcim

Full length article

Augmented reality spatial programming paradigm applied to end-user robot programming☆

Michal Kapinus *, Vítězslav Beran, Zdeněk Materna, Daniel Bambušek

Faculty of Information Technology, Brno University of Technology, Božetěchova 1/2, 612 00 Brno, Czech Republic

ARTICLE INFO

Keywords:
Robot programming
Augmented reality
SME future

ABSTRACT

The market of collaborative robots is thriving due to their increasing affordability. The ability to program a collaborative robot without requiring a highly skilled specialist would increase their spread even more. Visual programming is a prevalent contemporary approach for end-users on desktops or handheld devices, allowing them to define the program logic quickly and easily. However, separating the interface from the robot's task space makes defining spatial features difficult. At the same time, augmented reality can provide spatially situated interaction, which would solve the issue and allow end-users to intuitively program, adapt, and comprehend robotic programs that are inherently highly spatially linked to the real environment. Therefore, we have proposed Spatially Anchored Actions to address the problem of comprehension, programming, and adaptation of robotic programs by end-users, which is a form of visual programming in augmented reality. It uses semantic annotation of the environment and robot hand teaching to define spatially important points precisely. Individual program steps are created by attaching parametrizable, high-level actions to the points. Program flow is then defined by visually connecting individual actions. The interface is specifically designed for tablets, which provide a more immersive experience than phones and are more affordable and well-known by users than head-mounted displays. The realized prototype of a handheld AR user interface was compared against a commercially available desktop-based visual programming solution in a user study with 12 participants. According to the results, the novel interface significantly improves comprehension of pick and place-like programs, improves spatial information settings, and is more preferred by users than the existing tool.

1. Introduction

The Fourth Industrial Revolution (Industry 4.0) has drastically changed the established approaches to manufacturing, especially for small and medium-sized enterprises (SMEs). Tasks previously performed mainly by humans have, in recent years, started to be automated by collaborative robots (cobots). The number of cobots in the industry is increasing year by year and is expected to continue growing in the coming years. According to The Insight Partners [1], the Collaborative Robots Market was estimated to be worth $1.15 billion in 2022, and it is expected to reach $8.81 billion by 2030. SMEs are characterized by small batches and frequent changes in production, requiring frequent changes in robot programs. One way to reduce associated programming costs is to allow non-expert users to understand and adapt robot programs, which are typically mainly variations of pick and place tasks (palletizing, loading and unloading of machines, etc.).

Established programming techniques seem inapplicable as they suffer from low comprehensibility for non-expert users. A program is typically represented as a sequence of actions of several types, e.g., instructions for moving along a line, setting joint angles, or controlling an end effector position. These can be visualized as diagrams or just as lines of code. Just by looking at the representation of the program, it is difficult to match the actions to the actual steps in the program, i.e., "*Which of the 20 MOVE instructions is the one I am looking for?*" Moreover, in the context of SMEs, we believe that adapting an existing program will be a much more common scenario than creating a whole new program from scratch. When production changes slightly, the programmer will likely not bother setting up the robot and its program

☆ This paper was supported by the 5G-ERA project with funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 101016681.

* Corresponding author.
E-mail addresses: ikapinus@fit.vut.cz (M. Kapinus), beranv@fit.vut.cz (V. Beran), imaterna@fit.vut.cz (Z. Materna), bambusekd@fit.vut.cz (D. Bambušek).
URLs: https://www.fit.vut.cz/person/ikapinus/ (M. Kapinus), https://www.fit.vut.cz/person/beranv/ (V. Beran), https://www.fit.vut.cz/person/imaterna/ (Z. Materna), https://www.fit.vut.cz/person/bambusekd/ (D. Bambušek).

https://doi.org/10.1016/j.rcim.2024.102770
Received 10 October 2022; Received in revised form 14 February 2024; Accepted 28 March 2024
Available online 13 April 2024

all over again but rather duplicate an existing one and incorporate a few changes to suit the current batch. Therefore, fast and easy program comprehensibility is crucial, even for the authors of their own robot programs, because it is natural to forget one's own code as time passes.

In recent years, several programming tools for non-expert users have been introduced, such as [2–4] (more in Section 2), which are usually based on paradigms known from general programming, such as visual programming, or natural language processing. Several robot-specific paradigms, such as programming by demonstration or tangible programming, have been introduced but have not yet been deployed in the industry. From the existing approaches, we see visual programming as the one with the highest chance of being deployed in real applications; however, it has to be specifically adapted for the robotics context.

Robotic programs involve operations in a real environment. Therefore, clear and explicit visualization and manipulation of the spatial parameters of operations is an essential requirement for the programmer. Most of the current programming solutions have a weak link between the program representation and the real space. Spatial parameters are usually represented by a textual description (e.g., *coordinates: X = 0.5, Y = 0.9, Z = 1.8*) or visualized in a virtual workplace using a robot model. In both cases, the user must constantly switch their attention between the robot and the screen to mentally map the textual or virtual spatial representation to the real environment, which implies a high cognitive and attentional load [5].

Incorporating augmented reality (AR) allows the visualization of spatial information directly in the real environment, as has been demonstrated previously (e.g., [6]). Thus, a programming tool that displays the spatial parameters of robotic programs directly in the 3D space of the workplace can improve program understanding and reduce the need for context switching. However, the use of AR technology also brings difficulties. Current handheld AR suffers from inaccuracy in position estimation [7–9], which is a serious problem for robotic tasks requiring high precision. In addition, most AR devices introduce their own problems, such as a narrow field of view (head-mounted displays) or the need to hold the device in the hand (handheld devices).

When it comes to existing AR-based solutions, there are very few attempts to provide a general programming tool that would allow users to create and adapt programs for different tasks. An example could be [10], which displays a Blockly-like interface in 3D space, however, with a lot of textual information cluttering the interface. Most of the existing works focus on a specific task, such as teaching assembly process [4], trajectory specification [11,12] or allow only to set parameters to a pre-defined program [13].

We have specified five research objectives to address the aforementioned problems of contemporary solutions (commercially available end-user programming tools, which we further refer to as a *standard method*):

$RO_{comprehension}$ — Improve the robotic program comprehension over the standard method.

$RO_{simplicity}$ — The creation and adaptation of programs in AR should be at least as simple as in current end-user programming tools.

RO_{load} — Lower the perceived task load compared to the standard method.

$RO_{ergonomics}$ — Provide good handheld AR user interface ergonomics.

$RO_{precision}$ — Allow precise specification of spatial information using handheld AR.

We propose a new robot programming paradigm based on direct interaction with a virtual representation of the robot program visualized in a real environment using AR. The paradigm is based on the fundamental nature of manufacturing activity: (1) performing operations/actions, (2) in a specific position (or in motion), (3) with specific parameters, and (4) in some sequence of actions. Based on this reasoning, we define *Spatially Anchored Actions* (SAA). The spatial information of the program operations and their sequence are visualized in the real space of the robot by virtual 3D objects. The interface further benefits from semantic knowledge of the environment (manually or automatically acquired), i.e., the objects present and their models, and the user is provided with only contextually relevant menus. Knowledge of the positions and shapes of real objects also facilitates fast and accurate adjustment of the spatial parameters of the program. The method allows in-situ creation and visualization of the robotic program and interaction with virtual and real scene elements. Thus, the programmer works naturally in one environment (a mix of real and virtual) without the need for context switching.

The remainder of this paper is organized as follows: Section 2 presents the theoretical background of our method, which is explained and discussed in Section 3. Section 4 contains details of conducted user study and discovered observations, results of the user study are further discussed in Section 5, and everything is concluded in Section 6.

2. Related work

The increasing spread of cobots in the industry raises the demand for allowing end-users to program them. Naturally, robots can be programmed using a vendor-specific language, such as ABB's RAPID [14], Fanuc's Karel, or Universal Robots' URScript [15]. Although these languages offer relatively simple syntax and programming commands, they still require programmers with expertise in programming and robotics [16] and, therefore, are not suited for end-users.

To reduce programming complexity and to lower the requirements for the user's training level, some commercially available teach pendants offer a certain form of simplified programming. However, despite the effort to engage less experienced users, they often possess high mental and physical demands, lack the ability to use common syntax structures, and have no option for visualization [17]; therefore, their usability seems to be rather low [18]. Offline programming tools, such as ABB RobotStudio[1] [19], Fanuc RoboGuide[2] or RoboDK,[3] offer more functionalities and allow to program the robot in a simulated environment, which reduces the robot downtime, but on the other hand, requires extensive training and experience. Additionally, these desktop and pendant user interfaces imply a high cognitive and attention-related workload for the user due to a continuous switching of visual attention between the robot and the user interface [5].

Many alternative approaches for simplified robot programming have been proposed throughout the past years. To allow end-users to program robots, some used variations of visual programming [2, 20–22], programming by demonstration (PbD) [23], tangible programming [24,25], natural language interface [3], and some explored programming directly in the robot's space using AR [4,10–13,26,27]. The existing approaches also differ in the level of abstraction, where the lowest could be setting robot joints (very unintuitive and time-consuming), and the counterpart would be, for instance, programming on the task level, using PbD or reusable skills [28], which are better suited for end-users needs.

The published works usually differ in the type of device used for the interaction. Some of them used a head-mounted display (HMD) to program the robot by setting trajectory waypoints [11,26,27]; others used projected spatial AR [13], visual programming in combination with visualization of spatial waypoints in the workplace [10], or an HMD in combination with a handheld pointer [12]. Apart from robot programming, AR has been found useful for visualizations of robot programs and motions [29], inspection and maintenance [30], or training [31,32]. Moreover, AR can display the visual content directly in the working space, in one's line of sight, which reduces the user's cognitive load when switching the context and attention between the robot and an external device [33]. Based on the literature review and feedback

[1] new.abb.com/products/robotics/en/robotstudio.

[2] fanucamerica.com/products/robots/robot-simulation-software-FANUC-ROBOGUIDE.

[3] robodk.com.

from the industry, we see handheld devices as the most prospective choice for usage on a daily basis, as they are affordable and robust (which is important for industrial use-case), well known by users and, at the time, able to convey imperfect, but sufficient AR experience, providing an intuitive environment for in-situ programming.

Recently, frameworks such as Google ARCore[4] or Apple ARKit[5] enabled fast and easy development of AR applications for smartphones and tablets, which are in general significantly more affordable than HMD devices, and well known by users. Both deliver mandatory functionalities for AR using their closed-source implementation of Visual-Inertial Simultaneous Localization and Mapping [34–36]. The ARKit is generally faster than ARCore and reacts better to fast device movements [37]. Moreover, it has better results in localization and camera tracking, especially in industrial contexts and dynamic environments [38]. On the other hand, the ARCore is supported in both Android and iOS and can be used in many various devices from different manufacturers. With their current implementation, they are usable mainly for simple, small-scale environments [38] and non-complicated use-cases only, as hologram drifting can vary around 8 cm in challenging scenarios [7]. These AR frameworks are suitable for visualization and interaction tasks but not for the precise input of spatial information per se. If there is a need to input spatial information with high accuracy, AR should be combined with another technology, e.g., kinesthetic teaching, the approach we used for our interface.

To summarize, there exist various attempts to enable non-expert users, who might be domain experts, to program robots. AR seems to be a promising technique for this problem; however, until recently, it was limited to expensive devices (HMDs, SAR setups). With the availability of AR on consumer-grade handheld devices, new opportunities are opening up.

Our approach, the SAA method, extends existing visual programming approaches to 3D space, by which it attempts to allow end-users to work within their task space and eliminate the necessity to switch attention between a device and the workspace, which is inevitable in the majority of existing approaches. This is achieved by using AR, where the user interface is purposefully designed to be used on tablets, which are affordable for SMEs. The method overcomes the inherent lack of precision in AR tracking by using specifically designed graphical tools and leveraging robot precision. A program comprises high-level parametrizable actions inspired by skills and extended to be used in conjunction with AR. Within-workspace programming solves the main problem of contemporary solutions: a comprehension of programs that are highly spatially linked to the real environment. To the extent of our knowledge, this is the first solution enabling general visual programming in AR, with a user interface specifically designed for optimal ergonomy and prolonged usage on tablets.

3. Spatial programming paradigm

The two crucial parts of typical robot programming are the specification of individual program steps, i.e., what should happen, and the precise definition of spatial information, i.e., where it should happen. Depending on the programming method and selected level of abstraction, the first or latter (or both) could be derived automatically by the system (e.g., in imitation learning) or hidden from the user (e.g., when computer vision and robot motion planning are involved).

Both these parts are naturally related because most robotic actions use predefined or calculated coordinates. Many contemporary robot programming tools represent spatial data in a way that is not natural for non-experts, such as textual coordinates within the source code. For non-experts to understand the spatial dimension of a robotic program, more than just source code is required. When a 3D environment model is available, a visualization of important spatial parameters (points in space or robot trajectories) could be made. Unfortunately, the quality of the environment model heavily influences the visualization immersivity (low-quality models could be ambiguous or vague). Moreover, the visual representation of spatial information is usually separated from the action definition in the above-described example, as the visualization of waypoints occurs in a 3D scene in one window, and the source code is presented in another window. To understand the program and its spatial meaning, the programmer needs to merge these two pieces of information mentally.

Many robotic programs consist of just three types of instructions: move instructions, end-effector manipulation (open/close gripper, turn on/off suction), and IO control. However, the source code can be tough to read and understand without the proper naming of instructions or thorough comments. Some simulations using the 3D model of the environment could be utilized to overcome this problem. However, preparing such a simulation environment could be costly and time-consuming.

We define a novel spatial programming paradigm for handheld AR: Spatially Anchored Actions (SAA). The concept namely defines:

- The program as a sequence of the actions in the 3D space.
- The effective usage of AR for visualization and interaction with virtual objects in the 3D space.
- The interaction modes for seamless program editing in the 3D space on handheld devices.
- Methods for direct and indirect virtual object fast and precise manipulation in the 3D space.

The proposed paradigm is applied to a robot programming task that is a suitable scenario for it, and we use it to explain and test the paradigm. In this scenario, the SAA elements serve as anchors for actions (program steps), meaning that users can directly see where the individual actions of the program take place during the execution. The effective usage of AR is designed with respect to existing guidelines focusing on handheld device GUI ergonomy.

3.1. Spatially anchored actions

The proposed approach is based on flow diagrams and represents the robotic program as a sequence of individual actions connected to the program flow. *Anchored actions* represent the individual program steps (see Fig. 1). The *anchored actions* are connected using the *connections*, and in terms of flow diagrams, the *anchored actions* are nodes of the graph, representing the program, while the *connections* are the edges of the directed acyclic graph.

Each *action* is anchored to one of the *spatial anchors*, representing the spatial information, as stated above. Using the AR, the *spatial anchors* are rendered on the exact place where the *anchored action* will take place, i.e., the *action* intended to pick a cube is located above the mentioned cube. This concept combines the spatial meaning of programmed action with its spatial parameters, which is crucial for robotic programs. Moreover, a single *spatial anchor* could serve for more *actions*, simplifying modification of joint *actions* (such as objects picking and placing on the same spot) and potentially enhancing the program comprehension. The *spatial anchors* could be attached to *scene objects*, virtual counterparts of real objects in the scene. This enables the user to define a spatial parameter relative to the real objects.

The *spatial anchors* represent specific points or space poses. A simple sphere that is natural for the observer is sufficient to visualize a specific point. To visualize a pose, the model of the end-effector, with a specific orientation applied, could be used.

3.2. Interaction modes

The proposed user interface introduces interaction modes to enable fluent interaction with minimal interface overhead. Based on the current interaction mode, only relevant tools are available for the user so

[4] developers.google.com/ar.

[5] developer.apple.com/augmented-reality/arkit/.

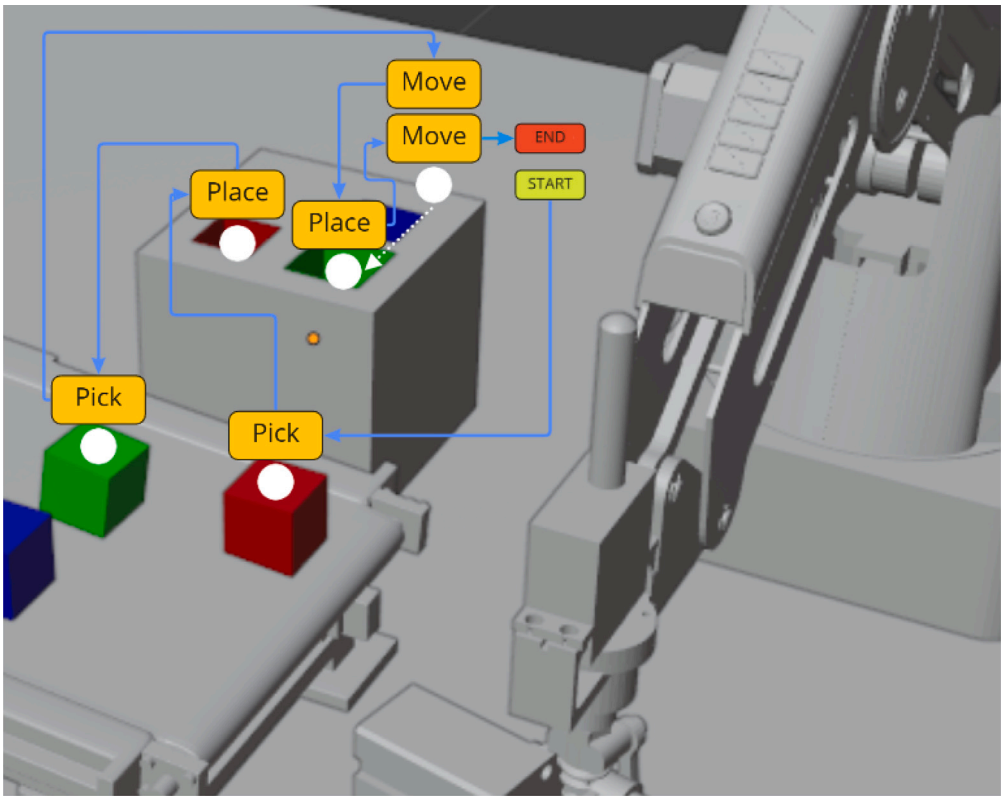


Fig. 1. The visualization of the *Spatially Anchored Actions* (SAA) concept. The white circles denote the *spatial anchors*, which serve to define and visualize spatial information. Above each *spatial anchor* is located one or more *actions*, represented by the yellow rectangle. The individual *actions* are connected by the blue lines, defining the program flow. Two *anchors* are connected by the white dashed line to show that the upper *anchor* is positioned relative to the lower *anchor*.

that they can focus on the current task and are not disturbed by an unnecessary on-screen interface. We propose five principal interaction modes.

The **execution mode** enables the user to execute selected *action*. The **transform mode** opens the transform menu over the selected *scene object* or *spatial anchor*. The **remove mode** enables the user to remove the selected *connection*, *action*, or *spatial anchor*. The **connection mode** allows the user to create arbitrary *connections* between two *actions*.

The **programming mode** allows the user to create program *actions* and *spatial anchors*. Its effects vary based on the selected object. When triggered, a context menu within the task space is opened, and the user can select desired *action* to be created. Once the *action* is selected, a new *spatial anchor* is created 20 cm from the tablet towards the front, and the *action* is attached to this *anchor*. Moreover, a *connection* is created automatically from the previous *action*. The *transform mode* is triggered afterward so that the user can specify the position of the new *spatial anchor*. The procedure differs based on the currently selected object:

- Existing spatial anchor: the new *action* is created and attached to the existing *spatial anchor*, and the *transform mode* is not triggered.
- Existing action: the new *action* is created and attached to the existing *spatial anchor* to which the selected *action* is attached, and the *transform mode* is not triggered.
- Scene object: the newly created *spatial anchor* is set relatively to the *scene object*, so when the user moves with the *scene object* (using the *transform mode*), the *spatial anchor* moves the same way.
- Connection: the newly created *action* is inserted in the program flow between the two *actions*, connected by the selected *connection*.

3.3. Ergonomy of the user interface

Most applications nowadays (including some AR/VR apps) use the *Windows, Icons, Menus, Pointer* (WIMP) paradigm to interact with the user. In AR applications, it usually means that most of the interaction is made using some "head-up" displays, which causes constant context switching between the scene and the display. To avoid this, we followed the design guidelines for UI elements in AR applications, as defined by the authors of ARCore framework.[6] The main outcomes for our user interface are:

- Move most of the interactive actions and feedback information directly in the scene to minimize the head-up interaction.
- Make the necessary interactive elements (buttons, sliders, etc.), which would be inconvenient to have in the scene, large enough, and place them in fixed, foreseeable places so they could be easily remembered and quickly reached without the need to look at them.
- Help the user to recover from missteps end errors by utilizing notifications displayed in the scene in front of the camera so the user sees it comfortably.

The proposed user interface's layout is presented on Fig. 2. It consists of three parts. The left part contains the main menu, allowing users to select one of the five interaction modes. The central part of the interface shows the scene image obtained from the camera. Additionally, a crosshair is placed in the middle of the screen, serving as a main virtual object selection tool. The right side contains two fixed buttons. The left one is the *mode button*, whose appearance and function differ based on the currently selected interaction mode. The right one

[6] https://developers.google.com/ar/design/interaction/ui.

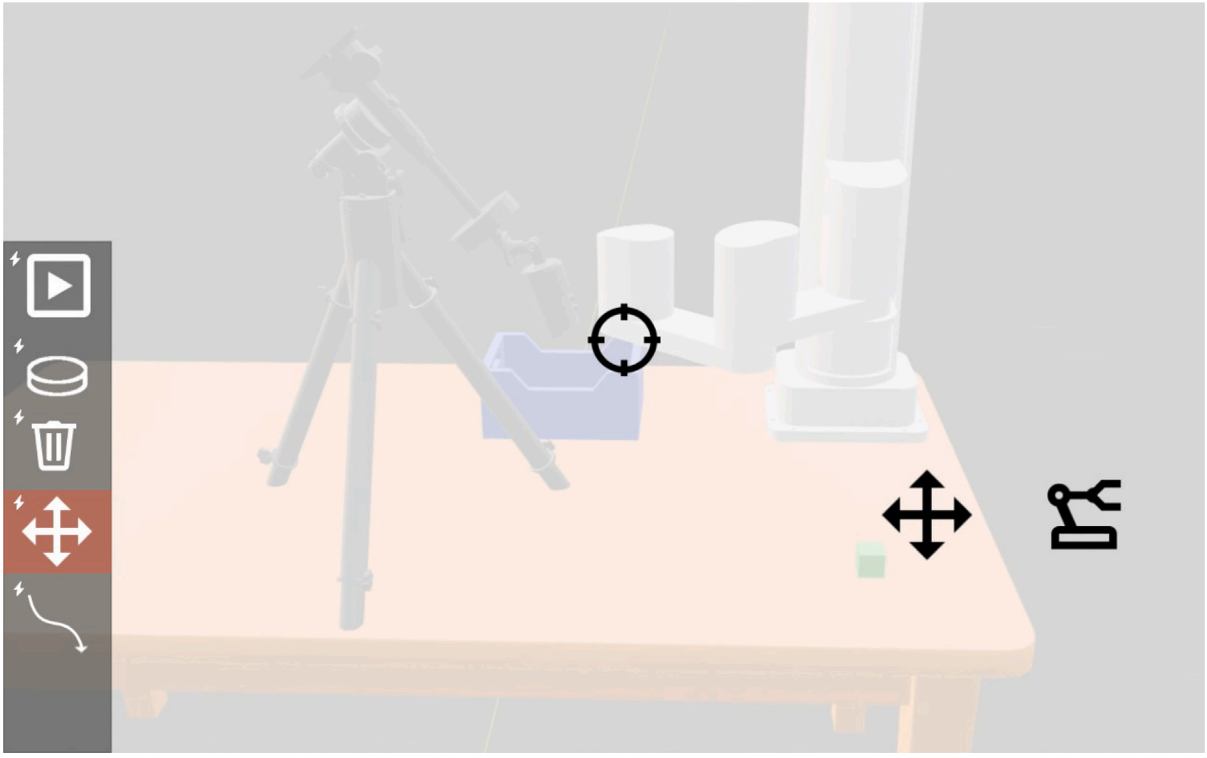


Fig. 2. Schematic visualization of the user interface. The left side contains the main menu, allowing users to select the appropriate interaction mode. In the middle is a crosshair for indirect virtual object selection. On the right side are two context-aware fixed *mode buttons*, easily reachable by the user's thumb.

relaxes the robot joints, allowing the operator to manipulate the robot arm. According to the abovementioned guidelines, both buttons are large enough and placed in the foreseeable place.

The buttons have no textual labels to save space and make the interface minimal. The help for each icon is shown upon the long button press, and a training session is expected before usage of the interface.

3.4. Precise programming in AR

The main drawback of using AR is the low accuracy of camera tracking when using standard devices (such as cell phones or tablets). In other words, using just an AR device to specify an exact point in space is virtually impossible, as the tracking error might reach lower tens of centimeters [7]. On the other hand, when it comes to robot programming, there is usually a very precise device available for point specification — the robot itself. The robot can be used to define points in space exactly. The problem with this approach lies in the visualization of the created program and the synchronization of the robot with other devices used in the program.

Our approach requires a workplace calibration to create a set of very precise **reference points**, which can be later used to specify spatial parameters precisely. These points can be defined by manual annotation using the robotic arm or automatically using computer vision technologies (e.g., QR code tracking). The reference points can be defined relative to the workplace or individual objects so they can automatically adapt to the objects' position changes.

Interaction widgets are used to precisely define several **relative points** using the imprecise AR visualization based on these reference points (see Fig. 1). The "parent" *anchor* is set using a precise method (manual guiding of robot in our prototype, but computer vision techniques can also be used). Other *anchors* are set using a combination of 2D and 3D widgets with selectable precision (see Fig. 3). We assume that, for understanding the program using its visualization in AR, the absolute precision (the correlation between the rendered virtual element and its actual position in the real environment) is not as important as the mutual relative precise position of virtual elements defining the program.

3.5. Transforming spatial anchors

The crucial interaction task is a manipulation with the *spatial anchor* in a real 3D task space. The proposed concept introduces direct (fast, but low precision) and indirect (slower, but precise) manipulation with the objects, i.e., *spatial anchors* or *scene objects*. Direct manipulation utilizes the physical movement of the handheld device. The *transform menu*, displayed on Fig. 3, contains a palm-shaped button for direct manipulation — when pressed, the object moves with the device's movement, allowing fast movement over large distances.

We propose an indirect manipulation for higher precision in setting the spatial parameters. The *rotary transform element* is placed on the right side of the *transform menu*, which allows moving the virtual object by scrolling the element. It clearly indicates the number of steps by which the object will be moved. The *magnitude selector* under the *rotary element* selects the length of the step. Together, it allows us to move the object by the exact distance. On the bottom are two buttons to change between the translation and rotation.

The user needs to see and select the direction in which the virtual object will be translated. We propose a 3D *gizmo* (see Fig. 3) for both cases. The *gizmo* consists of three perpendicular arrows representing the direction of the desired movement, and it is attached to the virtual object selected for manipulation. Close to the tip of each arrow, a current displacement from the original position is visualized. The desired direction of movement is indicated by selecting one of the arrows using the cross-hair.

Several buttons with additional functionality are in the left part of the *transform menu*. The arrows in the top serve for undo and redo operations. Under the palm-shaped button (described above) is the so-called *pivot* button. This button causes the object to move on the position of another object selected using the cross-hair. Using this button, the user can, for example, move a *spatial anchor* on the position

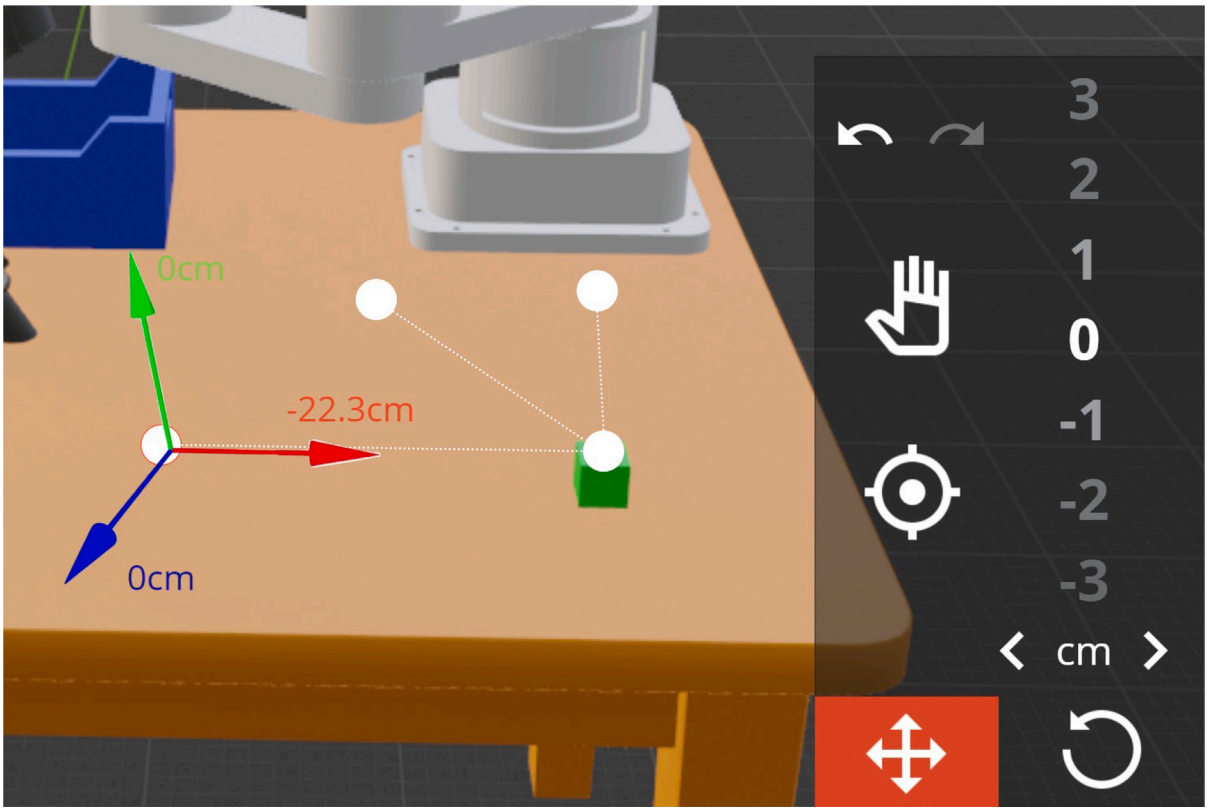


Fig. 3. The schematic visualization of the tools available in the *transform mode*. The left side contains the 3D widget (gizmo), rendered over the manipulated object. The right side contains the *transform menu*, with several interactive elements.

of aforementioned **reference point** and subsequently define a **relative point** using the *rotary element*.

4. Experimental evaluation

The method was implemented into a functional prototype, and a user study was carried out to compare it with a traditional approach for end-user programming on a 2D screen. The experiment was designed as a two-conditions within-subject user study, comparing the two different interfaces – our prototype interface based on presented SAA (C_{saa}) and the standard programming tool for the Dobot M1 robot – M1 Studio with the Blockly tool ($C_{blockly}$), which was selected as a representative example of a commercially available tool for end-user programming. We have stated four hypotheses related to the objectives above:

- $H_{comprehensibility}$ — The user is able to faster understand the program, seen for the first time, using the C_{saa} interface.
- $H_{usability}$ — The C_{saa} interface is more usable than $C_{blockly}$ and puts less task load on the user.
- H_{speed} — The user can create a new program faster using the C_{saa} interface than the $C_{blockly}$ interface.
- $H_{precision}$ — The C_{saa} interface provides similar precision for selected task as the $C_{blockly}$ interface.

The following chapter presents a user study we have prepared and conducted to test the hypotheses.

4.1. Prototype

A functional prototype[7] was prepared for the experimental evaluation, containing basic functionalities for programming of pick & place-like tasks. The prototype application was developed in the Unity3D game engine, using the AR Foundation framework,[8] which encapsulates the Google's ARCore,[9] for AR-related parts. The application is designed to run on Samsung Galaxy Tab S6 or S7, a 10″ Android tablet device compatible with the ARCore.

The prototype is designed as a non-immersive AR application, following the guidelines described in Section 3.3. The SAAs (see Fig. 4) are visualized as yellow arrows above blue spheres. The spheres represent spatial anchors, anchoring the actions for visualization and execution.

The prototype is fully functional, except for the object calibration procedure, which allows the user to set a precise object's position and orientation by navigating the robot's end-effector into several specific points on the object's body. In the experiment, this procedure was utilized to define the position of the workpiece. However, it was done using the Wizard of Oz approach, which was unknown to the participants. In the real-world scenario, functional robot-based calibration or calibration based on a computer vision technique would be used instead. Besides that, the participants interacted with a real, functional robot and created a robotic program from scratch.

4.2. Experiment design

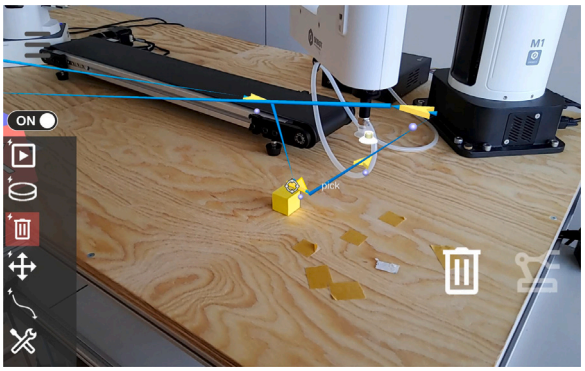
As was stated above, two conditions are studied: C_{saa} and $C_{blockly}$. Each condition represents an interface that utilizes visual programming and contains specialized elements for robot manipulation.

C_{saa} utilizes a custom handheld AR application for visual programming in task space based on the presented method of SAA described in the previous chapter. The participant was standing in front of the table and could interact with the workplace from the front and right side of the table (see Fig. 5(a)).

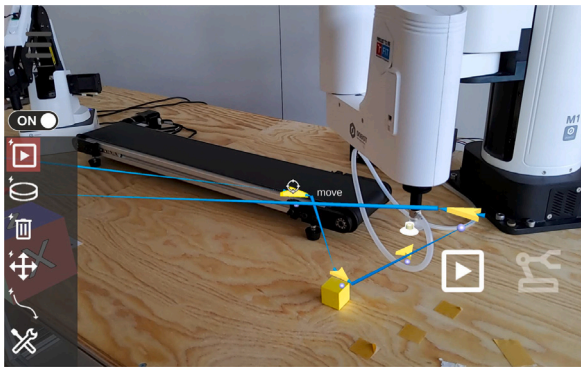
[7] Source code is available at github.com/robofit/arcor2_areditor.

[8] docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual.

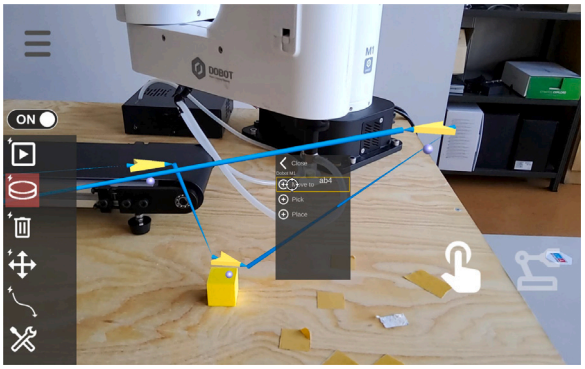
[9] developers.google.com/ar.



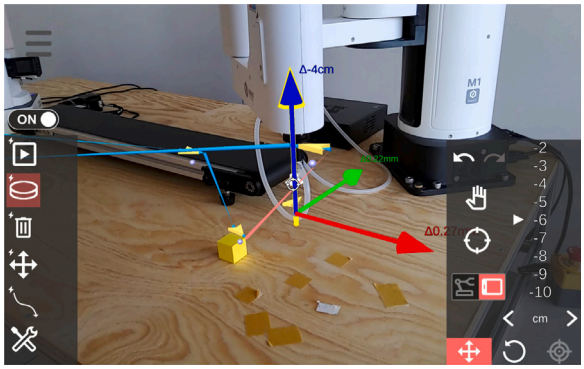
(a) The remove mode. The user could delete any virtual element by pressing the *mode button* when an object is selected.



(b) The execution mode. The user could execute any Action by pressing the *mode button* when an Action is selected.



(c) The programming mode. After pressing the *mode button*, an Action selector menu appears in front of the user, and they can select the Action to be created by selecting it with the crosshair and pressing the *mode button*.



(d) The transform mode. The transform menu is placed on the right side, allowing the user to manipulate the selected virtual object using the scrollable rotary element. The transform axis is selected using the crosshair on the transform gizmo (in the center of the screen). The gizmo shows the offset from the object's original position.

Fig. 4. Graphical user interface of the prototype application. The left side contains the main menu for mode selection. On the right side is placed either *mode button* (a–c), depending on the selected mode, or the transform menu (d) in case the object is being moved. The central part serves for viewing the scene with the superimposed interface.

$C_{blockly}$ uses an application for desktop computers with the Google Blockly framework for visual programming, where the user combines special puzzle-shaped boxes into a functional program. These blocks represent instructions such as `MoveJoints`, `SetArmOrientation`, etc. The parameters for each block are defined using either the keyboard or, in the case of move-blocks, by physical movement of the robot into the desired position. The participant was sitting on a chair by the table with a computer screen, mouse, and keyboard in front of the workplace (see Fig. 5(b)). They could reach the robot from the chair as well. They were allowed to stand up if they required better robot handling. The workplace was accessible from the front and right sides.

To minimize learning and transfer bias caused by the study being designed as a within-subject, the order of both conditions is randomized for each participant. For the safety purposes of both the robot and subjects, each participant was thoughtfully instructed on how to control the robots safely, the maximal velocity and acceleration of the robots were lowered to safe levels, and robots without sharp edges were selected for the study. The manipulated objects were small cubes made of foam to minimize the potential risk of injury.

4.3. Experiment protocol

Each experimental run was organized as follows. At first, the moderator welcomed the participant and asked them to sign an informed consent and fill in a demographic questionnaire. After that, a workplace introduction took place, and the moderator randomly assigned the first condition to the participant. After this introduction, the four experimental tasks were conducted.

Training task $T_{training}$

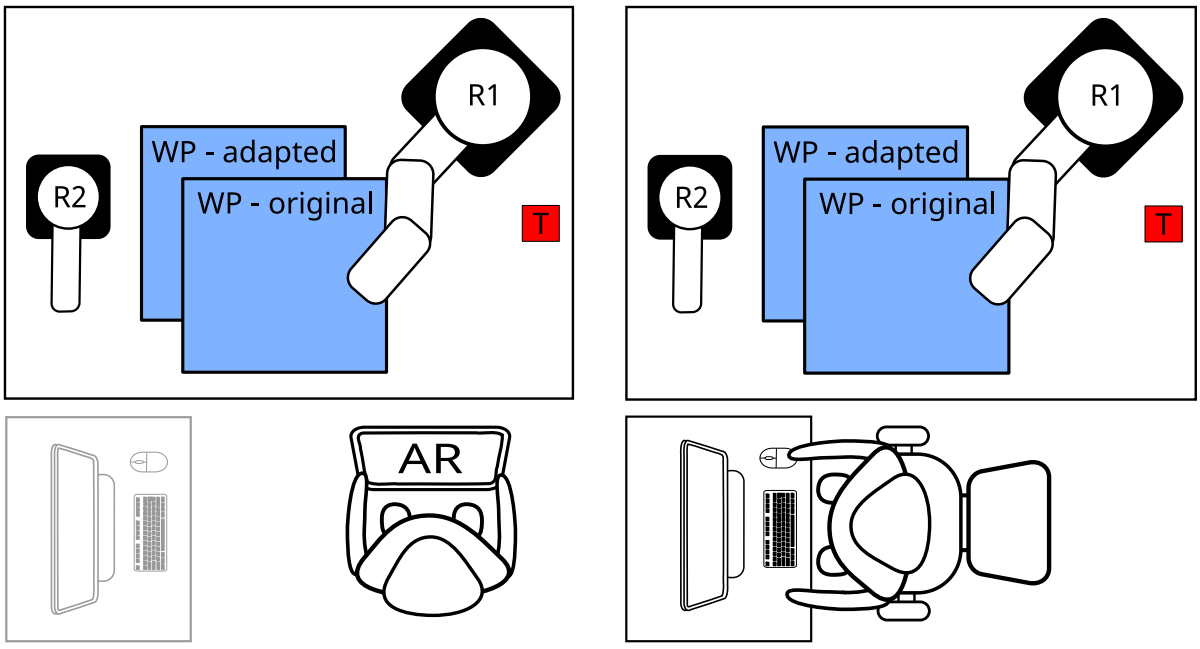
At first, the moderator introduced the participant to the programming tool and described the goal of the task. The participant was told to program the robot to pick a foam cube from the table and put it inside the box. The created program was to be subsequently modified, so the robot followed a specified path before the cube was released (the path was defined as a 10 cm line under the 45° angle, ending at a specific point on the bottom of the box). During this phase, the moderator proactively helped the participant with the programming, explained the required functionality, and answered all questions.

Visualization task $T_{visualization}$

An existing program was presented to the participant. Their task was to identify certain program steps according to the moderator's questions. The participant was explicitly informed that they could use anything the user interface offers, namely, the ability to run the program, program steps, or move the robotic arm. The presented program (see Fig. 6) differs for both conditions, so the participants were not influenced by previous knowledge of the presented program. Both programs involved the pick & place task with various objects, and the usage of the conveyor belt. All questions for both conditions are to be found below.

Questions for C_{saa}: Find the action, which causes…

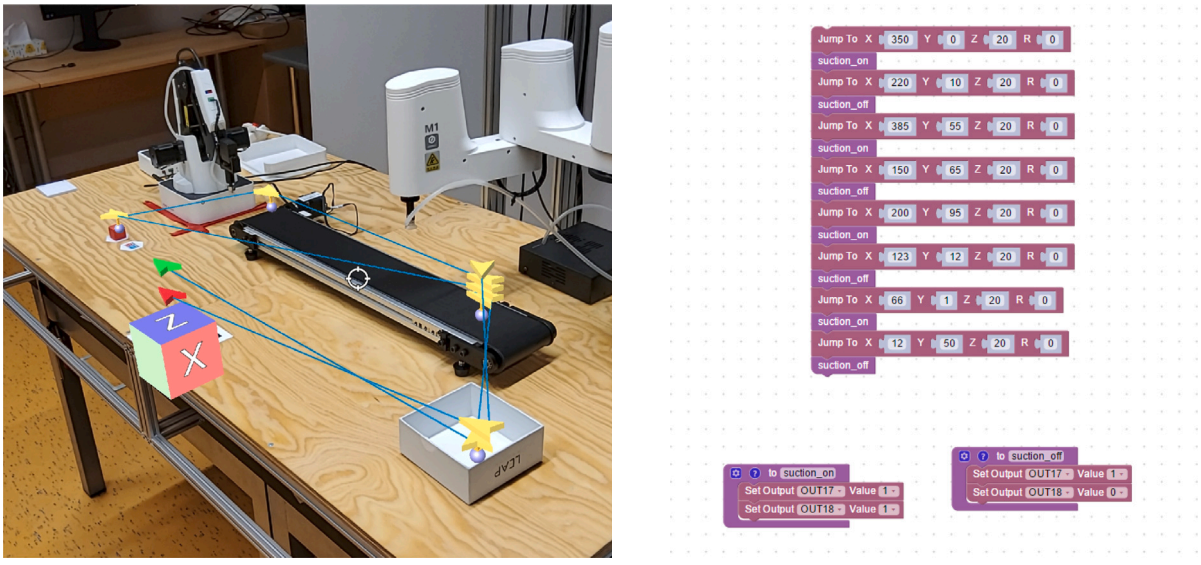
1. the bigger robot to pick the box from the conveyor belt.
2. the smaller robot to pick the cube from the table.
3. the conveyor belt to shift from the bigger robot to the smaller one.



(a) Scheme for C_{saa}. The person stands in front of the workplace and holds the tablet. The workplace is accessible from the front and the right side. The computer is present but not utilized in this condition

(b) Scheme for $C_{blockly}$. The person sits in front of the computer, which is located in front of the workplace. They can reach the robot from the chair as well.

Fig. 5. Workplace scheme for both conditions. The *R1* is the main robot, the Dobot M1. The *R2* is an additional robot, Dobot Magician, which was utilized only in the *visualization* task. The red square with the capital *T* is the original position of the object the *R1* should pick and manipulate. The blue squares represent the *workpiece* in two positions, the original and the adapted.



(a) The program for the C_{saa}. It consists of 8 actions that cause the big robot to move the box from the table to the conveyor belt and move it to the left. The smaller robot then picks up the red cube and places it in the box. The belt is then moved back to the right, and the bigger robot moves the box back to the table.

(b) The program for the $C_{blockly}$. It consists of 8 jump actions (type of movement) and four sets of suction_on and suction_off actions. Together, it causes the bigger robot to pick up four cubes of different colors from the table and place them in a new spot.

Fig. 6. The programs used for the visualization task $T_{visualization}$.

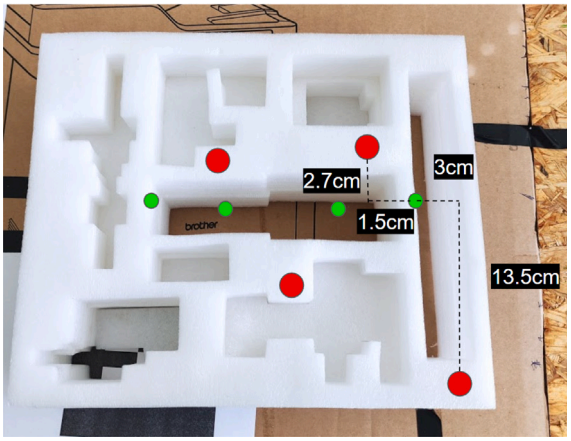
4. the bigger robot to pick the box from the table.

Questions for $C_{blockly}$: Find the action, which causes the bigger robot to...

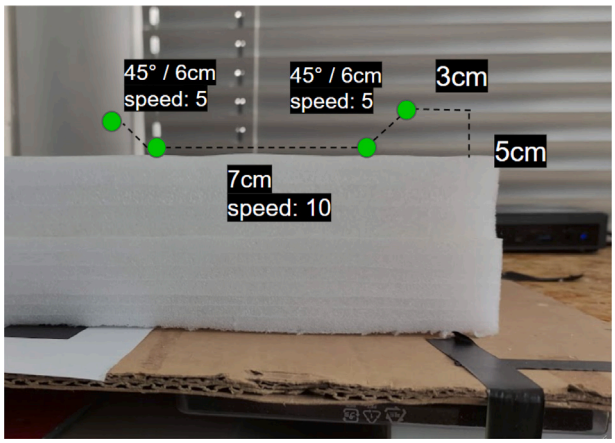
1. pick the yellow cube from the table.
2. place the red cube on the table.
3. move the green cube above the conveyor belt.
4. pick the blue cube from the table.

Main task T_{main}

The main task was presented to the participant. It simulates precise robotic manipulation with workpieces in a structured environment.



(a) The position of reference points (the large circles) and the trajectory points (the small circles). The first trajectory point's position is referenced with respect to the reference points.



(b) The trajectory points (circles), their mutual distances, and the speed at which the robot is intended to move between two consecutive points.

Fig. 7. The drawing of the intended trajectory for the main task, superimposed over the workpiece used for the experiment, from the top and front view. These drawings were available for the participants during the T_{main} and $T_{adaptation}$.

Specifically, the robot should pick a cube and perform simulated grinding by following a specific trajectory defined by a technical scheme (see Fig. 7), which was available to the participant during the session. The scheme contains the position of each waypoint and the speed of the end-effector's movement between two consecutive waypoints. Lastly, the robot should return the cube to the original table spot. The experiment task was the same for both conditions. For the C_{saa} condition, the participant had to annotate the position of the workpiece first as a part of the T_{main} so that they could utilize its reference points afterward. The procedure consisted of setting the position of four reference points on the workpiece (the red circles at Fig. 7(a)) using the hand movement of the robot. Once the annotation was done, the reference points were automatically added to the scene as spatial anchors. The participants were told to define other anchors relative to the reference points.

After the moderator answered the questions, the participant started to work. The participant was allowed to ask questions during this phase, and they were noted and categorized by the context of the question (i.e., if they were related to the task or the programming tool).

Adaptation task $T_{adaptation}$

The moderator moved the simulated workpiece to the new place, and the participant had to adapt the previously created program. In the case of C_{saa}, it meant only annotating the position of the workpiece again, as all related spatial anchors were defined relative to the workpiece's reference points. For the $C_{blockly}$, setting a new position for all the waypoints must be done again. For simplicity, the participants were told only to set the first waypoint.

Once all four tasks were done with the first condition, the participant was supposed to fill in questionnaires regarding the current condition. After that, the same procedure was conducted using the other condition. In the end, an open discussion took place. The moderator asked the participant for their impressions, additional questions, and opinions.

During all the tasks, the participant was allowed to test the execution of individual actions or the whole program. When the participant claimed that they thought the program was completed, the moderator observed and executed the program to check its functionality. In the case of problems, the moderator suggested what needed to be altered, and the participant was supposed to correct the program.

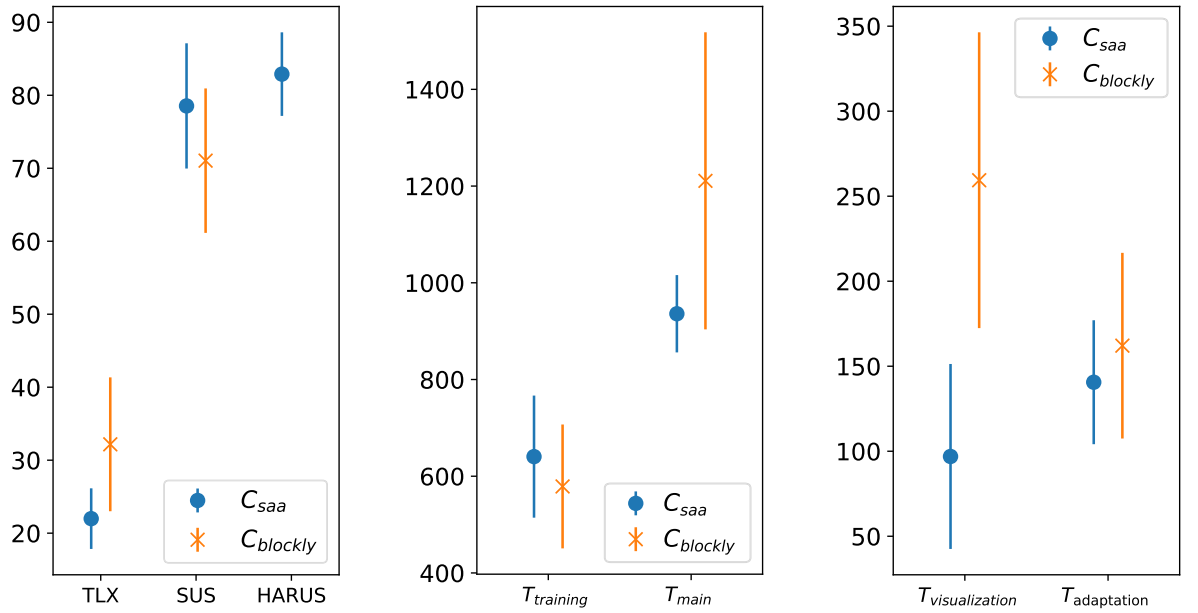
4.4. Dependent measures

As an objective measurement, the completion time was selected. This time is computed for each task separately so that we can compare the duration of each task individually for both conditions. The time reported only includes the time when the participant actively worked on the task. Intervals, where participants asked questions, a technical problem occurred, or the moderator had to intervene, were manually subtracted to report a pure task completion time. Asking the questions during the experiment can influence the participant's performance (and the completion time) under both conditions. However, because the performance can be influenced both positively and negatively (e.g., when the answer confuses the subject), we argue that the overall completion time is not significantly affected in either way.

As a subjective metric, standard questionnaires were selected. Namely, the NASA Task Load Index [39] for measuring mental and physical load, and the System Usability Scale [40] to rate the usability of the prototype interface. Besides these standard questionnaires, evaluated independently for each interface, another one containing specific questions regarding the prototype interface was utilized. Moreover, for the C_{saa} condition, the HARUS questionnaire [41], which is explicitly designed for the usability of handheld AR interfaces, was incorporated.

4.5. Participants

The user study was conducted with 12 subjects of various ages, self-reported genders, and technical backgrounds. Eleven participants identified themself as males; one identified themselves as female. Most subjects are shop-floor workers (6), students (2), or graduates from humanities colleges (2) with little or no prior experience in programming. One participant works as a programmer, and one works as a robot operator. They reported their experiences with robots on average 2.17 (on the scale of [1..5], where higher means more experienced), experiences with AR on average 2.25, and experiences with programming on average 2.08. Each participant signed informed consent to data recording and its usage for evaluation and eventually propagation in anonymized form. Some participants reported eye defects, such as myopia or amblyopia, but none reported that they affected them during the experiment. The user study took place in a lab-like environment in a dedicated room, where no external factors could influence the process of the experiment. All participants were able to finish all the tasks using both conditions.



(a) The subjective measurements. For the TLX, the lower means better. For SUS and HARUS, the higher means better.

(b) Time (in seconds) needed to complete $T_{training}$ and T_{main} (main) tasks.

(c) Time (in seconds) needed to complete the $T_{visualization}$ and $T_{adaptation}$ (adaptation) tasks.

Fig. 8. Comparison of subjective and objective measurements (mean values and corresponding 95 % confidence intervals) for conditions C_{saa} (proposed method) and $C_{blockly}$ (standard method).

5. Results

This section summarizes the user-study results and provides its analysis and interpretation. All statistical tests were done at the 5% significance level. Data were first tested for normality (combination of D'Agostino and Pearson's tests), and based on the result, paired t-test or Wilcoxon's signed-rank test were used to test for the significant difference between conditions.

5.1. Quantitative and qualitative data

Results from SUS and NASA-TLX questionnaires (shown in Fig. 8(a) for both conditions) show that, on average, the participants perceived a lower task load using the C_{saa} interface and ranked it as more usable. The mean TLX score for the proposed SAA interface (C_{saa}) was 21.99, which is less than 32.18 for the $C_{blockly}$. Regarding the usability of the interfaces for both conditions, the SUS questionnaire results show that participants consider the interface from C_{saa} more useful, scoring 78.54, compared to the $C_{blockly}$, scoring 71.04. However, differences are not significant for both metrics according to the paired t-test ($p = 0.074$ for TLX, $p = 0.312$ for SUS); therefore, the $H_{usability}$ cannot be confirmed. Besides, the C_{saa} scored 82.90 using the HARUS method, specifically designed to measure the usability of handheld AR systems. The score is higher than that of comparable interface SlidAR [42], which is aimed at virtual object manipulation and scored 76.3 (SD = 10.83).

The **training** time ($T_{training}$) was comparable for both interfaces, although longer with the C_{saa} interface (see Fig. 8(b)). Contrary, the **main task** (T_{main}) was significantly faster (including the time for calibrating the workpiece) with the C_{saa} interface according to the Wilcoxon test ($p = 0.042$); therefore, the H_{speed} was confirmed.

In the **adaptation** phase, the users were asked: for C_{saa} condition to complete the calibration procedure for the workpiece in the new position, and for $C_{blockly}$ condition to set the position of the first point of the trajectory. The completion times in Fig. 8(c) show that the adaptation using the C_{saa} condition was faster, although not significantly. On the other hand, in the $C_{blockly}$ condition, participants did not fit the entire trajectory, and therefore, the difference becomes even larger as the number of points on the trajectory increases.

Analyzing the completion times for the **visualization** ($T_{visualization}$), it was shown that for the C_{saa} condition, the participants required significantly less time to answer the questions (see Fig. 8(c)). This suggests that the AR interface supports the user in program comprehension, especially for the actions with the spatial information, which are crucial for robotic program understandability; therefore, the $H_{comprehensibility}$ is confirmed.

The discussion with the participants showed that they felt more certain when identifying the program steps using the SAA presented in AR. Identification of each step was instant just by looking over the scene and benefit from the fact that most of the program steps are represented by 3D objects placed on the spot where the action should take place. The only problem occurred when they had to identify the step causing the shift of the conveyor belt (third question within the C_{saa} condition), which has no clear spatial information. Two participants had difficulties in indentifying it, mainly because of the confusing textual description of the related action, and it took them 124 and 310 seconds respectively. The rest of the participants were able to identify the movement of the conveyor belt in a much shorter time with a mean of 25.5 (SD=13.9) seconds. With the $C_{blockly}$, users did not utilize the ability to run the program, although they were explicitly remarked that they might run it. Instead, they used the robotic arm to estimate the spatial coordinates of each program step to identify them. This strategy was successful but time-consuming. The participants, on average, needed 1.17 attempts (SD: 0.38) to identify the correct action for the $C_{blockly}$ interface and 1.2 attempts (SD: 0.45) for the C_{saa} interface, but over a significantly longer period of time.

5.2. Preferences

According to the C_{saa} interface questionnaire, eight out of twelve participants preferred the rotary control element for the precise movement of virtual objects. Two participants preferred the robot manipulation, and two preferred the free-form tablet motion. For the coordinates

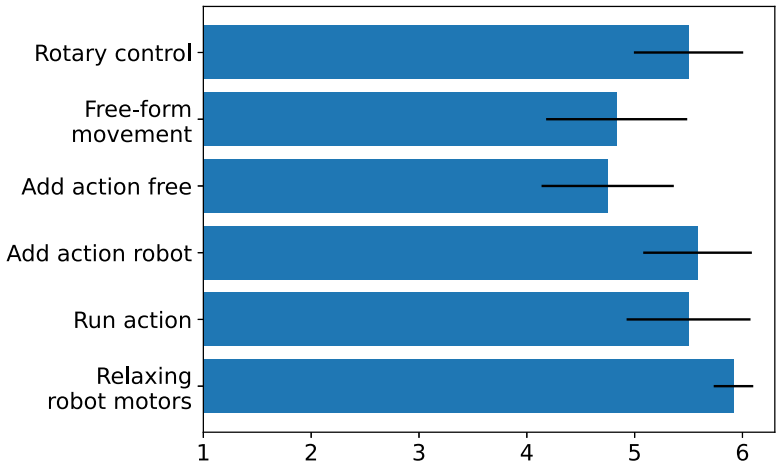


Fig. 9. Usefulness of selected features of C_{saa} interface, rated by the participants on a scale from 1 (useless) to 6 (very useful).

setting, where the approximate position is sufficient, half of the participants preferred the robot manipulator, a one third of participants preferred the free-form tablet motion setting, and only two of them preferred the rotary control element.

The **insertion of new spatial anchors** was preferred by the robot's end-effector rather than the free-form tablet position (see Fig. 9). We argue that this is because of a higher level of certainty, as the participants knew precisely where the spatial anchor would be placed and that the robot would be able to reach that position. **Setting a robot's end-effector path**, the participants usually followed the pattern: (1) setting a waypoint, (2) create a new waypoint on the position of the previous waypoint, (3) move the new waypoint in a certain direction. To achieve this pattern, the user had to create a new waypoint freely in the space (or at the position of the robot) and then use the *pivot* functionality (described in Section 3.5), which sets the position of the waypoint to another virtual object (previous waypoint in this case). According to our observations and discussion with the participants, they would appreciate the possibility of adding a new spatial anchor to an existing one, i.e., similar to adding it to the position of the robot's end effector. The robot motor's **unlock button** was considered a very useful tool by all of the participants (see Fig. 9).

5.3. Feedback and observations

There are three main categories of the feedback collected. The first deals with the tablet-based AR. Three participants expressed concern about the tablet falling out of their hands, especially when they were holding it with only one hand and controlling the robot with the other. One of them said, "*I'm afraid I will drop the tablet because I'm holding it with one hand, and I still have to hold down the release button. I would put it down on the table, but then again, I can't hold the button down properly*". Moreover, moving the robot with only one hand was also difficult for the participants. Three participants reported that at the beginning, they were stressed out by the C_{saa} interface, mainly because of a rich set of functions, compared to the $C_{blockly}$ interface. Moreover, they claimed that the 3D elements within the C_{saa} were entirely new to them, and it took some time for them to get used to it. Nevertheless, most of them agreed that after a short time, they got used to the controls, and the programming was easier than with the $C_{blockly}$ interface, despite their initial concerns: "*At the beginning it was a bit difficult to use the tablet, but once one understands the basic principle, it goes well*".

The second category deals with the program comprehension. For the task $T_{visualization}$, all but one participant (professional robot operator) preferred the C_{saa} interface. They claimed the spatial distribution of individual actions in task space helped to distinguish the *anchored actions*. One participant stated that they could quickly orient themselves because of the spatial visualization in C_{saa}. The other claimed that spatial visualization hugely helps them to identify which "pick" action is the one they are looking for, although they look the same: "*I liked the visualization better on the tablet, e.g., because of the naming of the actions and also that the position of the action in space made it easier to see which action was which*".

The third category of feedback deals with the comparison of both interfaces. Two participants preferred the programming using the $C_{blockly}$ interface over the C_{saa}. Both have a strong technical background; one works as a junior robot programmer (using RoboDK software), and the other has a background in CNC programming. The latter claimed that the visualization task was also easier for him using the $C_{blockly}$ interface: "*The visualization was better for me on PC — on the tablet, the actions were visible in space, but it seemed to me that on the PC I was more familiar with it*". Both stated that the $C_{blockly}$ interface was more straightforward for them and reminded them of the tools they were using at their jobs. The remaining participants preferred the C_{saa} for the programming tasks.

The participants generally liked the possibility of quickly executing *actions*, using the C_{saa} during the T_{main}, as it enabled them to check the reachability of the spatial anchors. With the $C_{blockly}$ interface, the participants were using the execution of individual actions more often, as they were using it also for identification of the actions in the programming tool, which was not needed in the C_{saa} interface because they saw the position of the spatial anchor in the AR.

Apart from the direct feedback collected through discussion with the participants, we have also collected some observations based on the recorded sessions.

All participants struggled with the visualization and control of the gizmo element in C_{saa}. They were often unsure which axis was selected or accidentally selected the wrong one. The participants struggled with the magnitude of the transform step selection, causing them to move the object at the wrong length or wonder why it was not moving because it only moved by several millimeters instead of centimeters. The transform widgets must be enhanced to provide better feedback for the operator on the desired movement's magnitude and direction.

Most participants considered the blue lines between the individual actions in the C_{saa} interface to be the robot's trajectory, although they were explicitly informed during the training that the blue line only indicates the order of the actions.

In the $C_{blockly}$, the users can modify the coordinates in textual form with virtually unlimited precision. The C_{saa} preserves the possibility of setting the position with a selectable degree of precision in a graphical way, utilizing the 2D and 3D widgets with user-defined coordinate systems. The participants finished all tasks using both interfaces, which required setting several precise spatial parameters. Therefore, we consider the $H_{precision}$ to be confirmed.

5.4. Limitations and future work

The 3D gizmo widget for axis selection was unclear for the participants as they were unsure which axis was selected, which was indicated by the thin outline of the selected axis. The currently selected distance/angle magnitude for transformation was also unclear, as it was indicated by the selection element in the lower right corner of the screen, thus outside of the user's view when working with the 3D gizmo. To check if the set spatial anchor is reachable by the robot, the participants had to execute an action attached to the anchor. It would be beneficial to visualize the reachability more clearly. We would also like to investigate more the possibilities for the robot motor's unlock button, as the dead-man-trigger concept causes trouble to the participants, forces them to hold the device in a non-ergonomic way, and causes trouble with robot manipulation. Moreover, we will evaluate the feasibility of the proposed concept in different contexts with no robots in action, such as home automation, where there is also a high demand for end-user programming techniques for definition of routines with IoT devices and, at the same time, a need to set spatial parameters as, e.g., the definition of various kinds of zones.

6. Conclusions

This paper presents a novel paradigm for spatial programming in handheld AR. The paradigm defines Spatially Anchored Actions for program visualization, their manipulation in real 3D space, and UI elements and rules for interaction in AR on handheld devices. The new concept was introduced and tested on a robot programming task. A fully functional prototype was created for tablet-like handheld devices, which was evaluated with 12 potential users and compared to the existing visual programming method. The study revealed that the SAA concept significantly helped the participant's comprehension and understandability of the robotic programs, which correlates with the research objective $RO_{comprehension}$. All participants successfully finished all tasks using both interfaces at a similar time; therefore, it was shown that the simplicity of program creation is similar to the standard tool ($RO_{simplicity}$). We also aimed to lower the users' task load (RO_{load}). The study revealed no significant task load reduction, which was relatively low for both tested conditions. A higher number of participants might be needed to reveal potential significant differences, as the variance in the data was rather high. One of our objectives was to provide good ergonomics for the handheld AR interface ($RO_{ergonomics}$). To do so, we have designed the user interface to be controlled by users' thumbs, enabling them to hold the tablet in an ergonomic position at the cost of requiring more thorough initial training.

Moreover, we designed the interface so it has most of the interaction elements located directly inside the 3D scene (instead of the traditional on-screen menus), allowing for lower context switching between the user interface and the visualization of the scene. We have also proposed several 2D and 3D widgets, allowing precise specification of spatial information using the AR ($RO_{precision}$). The users could finish the task with similar precision in both conditions.

CRediT authorship contribution statement

Michal Kapinus: Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Vítězslav Beran:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Zdeněk Materna:** Writing – review & editing, Validation, Software, Formal analysis, Data curation. **Daniel Bambušek:** Writing – review & editing, Software, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The datasets generated during the current study are not publicly available due to the participants' privacy but are available from the corresponding author on reasonable request.

References

[1] Insight Partners, Collaborative Robots Market Growth Report & Analysis by 2030, The Insight Partners, 2023, pp. 1–215.
[2] J. Huang, M. Cakmak, 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, IEEE, 2017, pp. 453–462.
[3] D. Fogli, L. Gargioni, G. Guida, F. Tampalini, A hybrid approach to user-oriented programming of collaborative robots, Robot. Comput.-Integr. Manuf. 73 (2022) 102234.
[4] S. Blankemeyer, R. Wiemann, L. Posniak, C. Pregizer, A. Raatz, Intuitive robot programming using augmented reality, Procedia CIRP 76 (2018) 155–160.
[5] A. Weiss, A. Huber, J. Minichberger, M. Ikeda, First application of robot teaching in an existing industry 4.0 environment: Does it really work? Societies 6 (2016) 20.
[6] M. Contero, J.M. Gomis, F. Naya, F. Albert, J. Martin-Gutierrez, Development of an augmented reality based remedial course to improve the spatial ability of engineering students, in: 2012 Frontiers in Education Conference Proceedings, 2012, pp. 1–5, http://dx.doi.org/10.1109/FIE.2012.6462312.
[7] T. Scargill, G. Premsankar, J. Chen, M. Gorlatova, Here to stay: A quantitative comparison of virtual object stability in markerless mobile ar, in: 2022 2nd International Workshop on Cyber-Physical-Human System Design and Implementation, CPHS, IEEE, 2022, pp. 24–29.
[8] A. Morar, M.A. Băluţoiu, A. Moldoveanu, F. Moldoveanu, A. Butean, V. Asavei, Evaluation of the arcore indoor localization technology, in: 2020 19th RoEduNet Conference: Networking in Education and Research, RoEduNet, IEEE, 2020, pp. 1–5.
[9] E. Battegazzorre, D. Calandra, F. Strada, A. Bottino, F. Lamberti, Evaluating the suitability of several ar devices and tools for industrial applications, in: International Conference on Augmented Reality, Virtual Reality and Computer Graphics, Springer, 2020, pp. 248–267.
[10] E. Yigitbas, I. Jovanovikj, G. Engels, Simplifying robot programming using augmented reality and end-user development, in: IFIP Conference on Human-Computer Interaction, Springer, 2021, pp. 631–651.
[11] C.P. Quintero, S. Li, M.K. Pan, W.P. Chan, H.M. Van der Loos, 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.
[12] S.K. Ong, A. Yew, N.K. Thanigaivel, A.Y. Nee, Augmented reality-assisted robot programming system for industrial applications, Robot. Comput.-Integr. Manuf. 61 (2020) 101820.
[13] Z. Materna, M. Kapinus, V. Beran, P. Smrž, P. Zemčík, Interactive spatial augmented reality in collaborative robot programming: User experience evaluation, in: 2018 27th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN, IEEE, 2018, pp. 80–87.
[14] R. ABB, Technical Reference Manual: Rapid Instructions, Functions and Data Types, ABB Robotics, 2014.
[15] U. Robot, The Urscript Programming Language for E-Series, Universal Robot, 2022.
[16] G. Ajaykumar, M. Steele, C.M. Huang, A survey on end-user robot programming, ACM Comput. Surv. 54 (2021) http://dx.doi.org/10.1145/3466819.
[17] G. Ajaykumar, C.M. Huang, User needs and design opportunities in end-user robot programming, in: Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction, 2020, pp. 93–95.
[18] C. Schmidbauer, T. Komenda, S. Schlund, Teaching cobots in learning factories–user and usability-driven implications, Procedia Manuf. 45 (2020) 398–404.
[19] C. Connolly, Technology and applications of abb robotstudio, Ind. Robot: Int. J. (2009).
[20] Y. Gao, C.M. Huang, Pati: a projection-based augmented table-top interface for robot programming, in: Proceedings of the 24th International Conference on Intelligent User Interfaces, 2019, pp. 345–355.
[21] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, G.D. Hager, Costar: Instructing collaborative robots with behavior trees and vision, in: 2017 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, pp. 564–571.
[22] C. Mayr-Dorn, M. Winterer, C. Salomon, D. Hohensinger, R. Ramler, Considerations for using block-based languages for industrial robot programming-a case study, in: 2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering, RoSE, IEEE, 2021, pp. 5–12.
[23] S. Alexandrova, M. Cakmak, K. Hsiao, L. Takayama, Robot programming by demonstration with interactive action visualizations, in: Robotics: Science and Systems, Citeseer, 2014, pp. 48–56.

[24] Y.S. Sefidgar, P. Agarwal, M. Cakmak, Situated tangible robot programming, in: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction, HRI, IEEE, 2017, pp. 473–482.
[25] Y.S. Sefidgar, T. Weng, H. Harvey, S. Elliott, M. Cakmak, Robotist: Interactive situated tangible robot programming, in: Proceedings of the Symposium on Spatial User Interaction, 2018, pp. 141–149.
[26] S.Y. Gadre, E. Rosen, G. Chien, E. Phillips, S. Tellex, G. Konidaris, End-user robot programming using mixed reality, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 2707–2713.
[27] M. Ostanin, A. Klimchik, Interactive robot programing using mixed reality, IFAC-PapersOnLine 51 (2018) 50–55.
[28] J. Hoyos, A.B. Junaid, M.R. Afzal, A. Tirmizi, P. Leconte, Skill-based easy programming interface for industrial applications, in: 2022 IEEE/SICE International Symposium on System Integration, SII, IEEE, 2022, pp. 210–217.
[29] E. Rosen, D. Whitney, E. Phillips, G. Chien, J. Tompkin, G. Konidaris, S. Tellex, Communicating robot arm motion intent through mixed reality head-mounted displays, in: N.M. Amato, G. Hager, S. Thomas, M. Torres-Torriti (Eds.), Robotics Research, Springer International Publishing, Cham, 2020, pp. 301–316.
[30] H. Eschen, T. Kötter, R. Rodeck, M. Harnisch, T. Schüppstuhl, Augmented and virtual reality for inspection and maintenance processes in the aviation industry, Procedia Manuf. 19 (2018) 156–163.
[31] E.Z. Barsom, M. Graafland, M.P. Schijven, Systematic review on the effectiveness of augmented reality applications in medical training, Surg. Endosc. 30 (2016) 4174–4183.
[32] S. Werrlich, K. Nitsche, G. Notni, Demand analysis for an augmented reality based assembly training, in: Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments, 2017, pp. 416–422.
[33] R. Suzuki, A. Karim, T. Xia, H. Hedayati, N. Marquardt, Augmented reality and robotics: A survey and taxonomy for ar-enhanced human–robot interaction and robotic interfaces, in: CHI Conference on Human Factors in Computing Systems, 2022, pp. 1–33.
[34] H. Liu, M. Chen, G. Zhang, H. Bao, Y. Bao, Ice-ba: Incremental, consistent and efficient bundle adjustment for visual-inertial slam, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1974–1982.
[35] T. Taketomi, H. Uchiyama, S. Ikeda, Visual slam algorithms: A survey from 2010 to 2016, IPSJ Trans. Comput. Vis. Appl. 9 (2017) 1–11.
[36] T. Terashima, O. Hasegawa, A visual-slam for first person vision and mobile robots, in: 2017 Fifteenth IAPR International Conference on Machine Vision Applications, MVA, IEEE, 2017, pp. 73–76.
[37] P. Nowacki, M. Woda, Capabilities of arcore and arkit platforms for ar/vr applications, in: International Conference on Dependability and Complex Systems, Springer, 2019, pp. 358–370.
[38] T. Feigl, A. Porada, S. Steiner, C. Löffler, C. Mutschler, M. Philippsen, Localization limitations of arcore, arkit, and hololens in dynamic large-scale industry environments, in: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP, INSTICC. SciTePress, 2020, pp. 307–318, http://dx.doi.org/10.5220/0008989903070318.
[39] S.G. Hart, L.E. Staveland, Development of nasa-tlx (task load index): Results of empirical and theoretical research, Adv. Psychol. 52 (1988) 139–183.
[40] J. Brooke, et al., Sus-a quick and dirty usability scale, in: Usability Evaluation in Industry, Vol. 189, 1996, pp. 4–7.
[41] M.E. Santos, J. Polvi, T. Taketomi, G. Yamamoto, C. Sandor, H. Kato, A usability scale for handheld augmented reality, 2014, http://dx.doi.org/10.1145/2671015.2671019.
[42] J. Polvi, T. Taketomi, G. Yamamoto, A. Dey, C. Sandor, H. Kato, Slidar: A 3d positioning method for slam-based handheld augmented reality, Comput. Graph. 55 (2016) 33–43, http://dx.doi.org/10.1016/j.cag.2015.10.013, URL: https://www.sciencedirect.com/science/article/pii/S0097849315001806.

Michal Kapinus, Ph.D. is a researcher at the Faculty of Information Technology, Brno University of Technology, Czech Republic, where he successfully defended his doctoral thesis “End-user Robot Programming using Augmented Reality”. His research interests are mainly human–machine interaction in mixed/augmented reality and end-user robot programming.

Vítězslav Beran, Ph.D. is an associate professor at the Faculty of Information Technology, Brno University of Technology, Czech Republic, where he leads the Human–Robot Interaction group. His research interests include human–machine interaction, computer vision, video processing and augmented reality. He has participated in several European and contractual research projects.

Zdeněk Materna, Ph.D. received a bachelor’s degree in computer systems in 2009 at the College of Polytechnics Jihlava. Then at the Brno University of Technology, he received a master’s degree in cybernetics, control, and measurements in 2011 and later, in 2019, defended a dissertation on the topic of advanced task-oriented user interfaces for non-expert users. His research interests are human–robot interaction, augmented reality, semi-autonomous systems, and intelligent home automation. He is currently a post-doc at BUT.

Daniel Bambušek is a Ph.D. student at the Faculty of Information Technology, Brno University of Technology, Czech Republic, where he focuses on the use of augmented reality in the field of human–robot interaction, including communication of spatial information and end-user robot programming. He is also interested in augmented reality and augmented virtuality for efficient user interfaces of drone operators.