END-USER COBOT PROGRAMMING IN AUGMENTED REALITY

PROGRAMOVÁNÍ KOBOTŮ V ROZŠÍŘENÉ REALITĚ KONCOVÝMI UŽIVATELI

DOCTORAL THESIS
DISERTAČNÍ PRÁCE

AUTHOR ING. MICHAL KAPINUS
AUTOR PRÁCE

SUPERVISOR ING. VÍTĚZSLAV BERAN, PH.D.
VEDOUCÍ PRÁCE

BRNO 2022

Ing. Michal Kapinus: *End-user cobot programming in Augmented Reality*, doctoral thesis Brno, Brno University of Technology, Faculty of Information Technology, 2022.

I would like to dedicate this thesis to my beloved wife,
who tolerate the fact that I'm just a boy in an adult's body.

ABSTRACT

Due to the recent expansion of affordable, collaborative robots, a rapid increase comes in the automation share in small and medium enterprises, and it is expected to be even higher in upcoming years. Together with the increasing number of robots, the number of robot programmers must inevitably rise as well. To boost the spread of automation, the price for robot programming needs to be decreased; otherwise, small enterprises would not be able to afford it. One way to decrease the programming cost is to simplify the process and allow less-skilled and less-educated operators to perform it.

The presented thesis deals with the problem of contemporary user interfaces for end-user robot programming, which are still too demanding for ordinary shop-floor workers. As for now, most of the interaction between humans and robots, both during the programming and the execution phases, takes place, not in the space occupied by the robot but somewhere else – in a computer, on the display of the attached device, etc. This forces the human to think of spatial relations of the created program constantly and to map the program to the real environment mentally.

To overcome this problem, the thesis proposes several methods based on Augmented Reality, which presents the spatial information in the actual environment in an understandable way. Moreover, in-situ user interfaces for such interaction methods are presented and evaluated in several user studies with more than 70 participants. A fully functional prototype of a universal robot programming tool intended for the end-users is presented.

KEYWORDS

Augmented Reality; End-User Robot Programming; Human-Robot Interaction, Natural Interfaces.

ABSTRAKT

Vzhledem k současnému rozmachu dostupných kolaborativních robotů pomalu nastává rapidní zvyšování podílu automatizace v malých a středních podnicích a je očekáváno, že toto zvyšování dále poroste v nadcházejících letech. Společně se zvyšujícím se počtem robotů, musí nevyhnutelně růst i počet programátorů těchto robotů. Aby ke zvyšování podílu automatizace mohlo dojít, cena za programování robotů musí být snížena. V opačném případě by si malé podniky nemohly automatizaci dovolit. Jeden ze způsobů, jak tuto cenu snížit, je zjednodušení programovacího procesu a zpřístupnění programování i méně školeným zaměstnancům.

Předložená práce se zabývá problémem současných uživatelských rozhraní, se zaměřením na rozhraní pro zjednodušení programování robotů, které jsou stále příliš náročné pro běžné dělníky. V současnosti, většina interakce mezi člověkem a robotem, jak v průběhu programování, tak v průběhu vykonávání programu, se odehrává jinde než v prostoru, který daný robot přirozeně obývá – na počítači, na obrazovce nějakého přídavného zařízení a podobně. Tento fakt nutí člověka neustále přemýšlet o prostorových náležitostech tvořeného programu a mentálně mapovat daný program na reálné prostředí.

Abychom překonali tento problém, předložená práce nabízí několik metod založených na Rozšířené Realitě, které prezentují prostorové informace ve skutečném prostředí ve srozumitelné formě. Kromě toho, společně s danými interakčními metodami jsou představeny také uživatelská rozhraní jež tyto metody implementují, a které jsou vyhodnoceny v uživatelských studiích s více než 70 účastníky. Práce také představuje plně funkční prototyp univerzálního robotického programovacího nástroje určeného pro koncové uživatele.

KLÍČOVÁ SLOVA

Rozšířená realita; Programování robotů koncovými uživateli; Interakce mezi člověkem a robotem, Přirozená rozhraní.

BIBLIOGRAPHIC CITATION

Ing. Michal Kapinus: *End-user cobot programming in Augmented Reality*, doctoral thesis Brno, Brno University of Technology, Faculty of Information Technology, 2022.

DECLARATION

I declare that this dissertation thesis is my original work and that I have written it under the guidance of Ing. Vítězslav Beran, Ph.D. All sources and literature that I have used during my work on the thesis are correctly cited with complete reference to the respective sources.

Brno, 2022

Ing. Michal Kapinus,
August 31, 2022

ACKNOWLEDGMENTS

Firstly, I would like to thank my parents and my family for raising me the way they did. I want to thank my beloved wife, who always supported me during both the light and dark phases of my Ph.D. study. Many thanks also go to my children, Šimon and Natálie, who were born during my Ph.D. study and who know how to make me smile and how to make me cry. I would like to thank all my current and former colleagues from our office, namely Zdeněk Materna, Daniel Bambušek, Martin Veľas, and Alfredo Plascencia. They provided me with countless brainstorming sessions, encouraging comments, and an incredible amount of fun, all of which were necessary to develop this thesis. I would also like to thank all of the other members of the department of computer graphics, as they all are great colleagues, and the department and faculty itself, for providing me with a beautiful workplace and the necessary funds to feed my family. Lastly, I would like to thank my supervisor, Vít'a Beran, for being not only a good supervisor but for being a great friend as well.

Two research projects supported the work on my thesis: *Collaborative robot 2.0: cognition of the work environment, AR-based user interface, simple deployment, and reconfiguration* and *Test-it-off: Robotic offline product testing*. Both projects allowed me to research different aspects of using augmented reality in end-user robot programming scenarios. The former using a projected spatial augmented reality and the latter using the augmented reality on a mobile device.

The template for this thesis was generously provided by Lukáš Polok.

CONTENTS

LIST OF ACRONYMS

AR	Augmented Reality	1
VR	Virtual Reality	2
MR	Mixed Reality	2
SAR	Spatial Augmented Reality	3
SME	Small and Medium Enterprises	8
PbD	Programming by Demonstration	11
HMD	Head-Mounted Display	12
PCB	Printed Circuit Board	19
WoZ	Wizard-of-Oz	20
SAA	Spatially Anchored Actions	24
SUS	System Usability Scale	39
NASA-TLX	NASA Task Load Index	39
HRI	Human-Robot Interaction	39
UEQ	User Experience Questionnaire	51
HARUS	Handheld Augmented Reality Usability Score	94
HCI	Human-Computer Interaction	165

1 INTRODUCTION

Human and Computer. Two completely different things, yet, every day, forced to interact with each other. Humans are emotional beings, used to living in a dynamic world, using imagination and communicating daily with other emotional beings. On the other hand, computers are strict, cold, and logical devices focused on working precisely at high speed without any emotions. So how can they interact with each other in the way that computers correctly understand human orders and, at the same time, humans can observe computers' work in appropriate form? The answer seems simple: a computer screen, keyboard, mouse, voice, printers, graphical interfaces, and much more. However, is the answer that simple for all cases? It is, and it is not.

To see the computer world from a human perspective means, in most cases, removing one dimension and presenting information on a flat, two-dimensional screen. Reading documents in this form is natural, just like reading a sheet of paper in the real world. In contrast, modeling a 3D object on a computer differs dramatically from what a sculptor does when forming a statue. Using a 3D editor like Blender or CAD software, when the user wants to see the object from the opposite side, they cannot just turn their head or walk around; they need to use a mouse to rotate the scene, which results in movement of the virtual camera and re-rendering of the flat image. Does it seem to be natural? Maybe it does, because everybody has been using it for the past 40 years. However, what is the cause, and what is the effect? Does everyone use it because it is great, or do we consider it good because there is no alternative option? These are the questions I will try to answer in the presented thesis.

Our long-term research objective is to enable end-users to program collaborative robots in a simplified way. I have focused my research on the possibilities of Augmented Reality (AR) for visualization and manipulation of digital data in the robot's task space (i.e., the space, where the task is performed) in the real world. Moving the interaction from the computer screen to the task space could increase the interaction effectivity. I have selected the end-user collaborative robots' programming as a representative example where bringing the interaction to the task space could significantly impact the interaction's effectiveness and usefulness.

The robots manipulate the objects in the real world. They can move tools and workpieces, drill holes into wooden planks, and do other manipulations with the environment. Although they manipulate the real world, when we instruct them or observe their behavior, we use virtual representation on the flat computer screen in most cases. Two basic robot programming methods are so-called offline programming and online programming. The former means that the robot program is created independent of the actual robot or robotic cell [37]. It could benefit from the knowledge of the CAD representation of the robot and workpiece and achieve high accuracy. Most robot programs follow stored waypoints with defined transitions between them with the occasional change of the end-effector state, i.e., opening or closing of the gripper or turning on and off the suction. The programming tools utilize the robot programming language, which could be either textual or graphical. In the case of the textual language, the programmer types different instructions in the file. In the case of visual language, the programmer assembles different visual pieces, e.g., blocks or puzzle pieces, to create the program. In both cases, the user sits behind the computer, works with a mouse and keyboard, and observes the digital information on the computer screen; therefore, the user must constantly map the virtual environment to the real one, similarly to spatial knowledge acquisition in navigation problems, which is ineffective [61].

In the case of online programming, the programmer typically works directly with the robot [37]. They can move the robot with their own hands or through the controls on the teach pendant and set the spatial parameters in the task space. Nevertheless, apart from setting the spatial parameters, the interaction occurs only through some virtual elements on the artificial device; the teach pendant or a computer. The only possibility for the robot to visualize the program is to perform it, which could be slow and unnecessary in some cases. The program visualization, therefore, occurs on the screen.

To bring the interaction into the robot's task space, selecting appropriate technology that provides visualization and control methods applicable in the actual environment is necessary. We need to mix the reality, i.e., the workplace, the workpiece, the robot itself, and the digital world, i.e., the representation of the program, the spatial information of the program, and the robot's state. A common term for mixing these two worlds is the Mixed Reality (MR) [133]. Two integral parts (among others) of the MR are the so-called Virtual Reality (VR) and AR. The former states for a fully immersive environment, where all content the user sees is purely virtual and fully synthetic [133]. On the other hand, the AR states for superimposition of virtual content into the real-world view [133]. The AR offers visualization and interaction methods suitable for the real-world environment. Therefore, we have selected it as the leading technology for *bringing the interaction to the task space*.

This thesis presents two AR-based approaches to instruct the robot on what to do and visualize the robot's program and current state. Both approaches deal with interaction in the robot's task space. The Part i of the thesis presents the necessary overview of contemporary approaches, describes our preliminary research, and presents the thesis statement. The Part ii deals with a projected Spatial Augmented Reality (SAR) and its usage to set the robot program's parameters. The mobile AR is utilized in the Part iii. The basic concept is presented first, followed by the description of conducted experiments with a functional prototype. Lastly, the Part iv describes my research outcomes and discusses possible future extensions.

Part I

PRELIMINARY RESEARCH AND THESIS GOAL

The following chapters show why I think bringing an interaction to the task space is a good idea and how it could be done using augmented reality. The reader can see the motivation behind it and why end-user robot programming is an ideal use case of such an interaction paradigm.

2 MOTIVATION AND OVERVIEW

Robots can replace humans doing repetitive or dangerous tasks or tasks requiring high precision. They can operate in a hazardous environment and handle heavy loads or toxic substances. On the other hand, they cannot work side to side with humans due to their lack of perception, high movement speed, and enormous forces. The robots must be separated from humans using physical or sensory barriers, such as safety fences or laser barriers.

Collaborative robots, or cobots, emerged to remove these limitations. They operate at lower speeds and implement other precautions to work alongside humans safely. The cobots typically perform simple and repetitive tasks, such as loading the workpiece into various machines like a press, lathe, or milling cutter, followed by their unloading once the operation is finished. Another common task is polishing, spraying, or cleaning surfaces, primarily because of their ability to imitate and repeat movement patterns learned by humans, which will be discussed in the following chapters.

The advantage of cobots is relatively easy integration into the existing factories, as they are usually compact and able to interact with their surroundings. For example, when the task is loading a workpiece inside the press, a button is pressed at a particular moment so the press may start the operation. The cobot usually has not to be electrically connected to the press machine as a standard robot, but it can just physically press the button, just like humans do. It can also open or close openings of various machines using its arm, which once again reduces the need for a change in the environment.

Despite the versatility of cobots, it is unlikely that they will replace all humans in enterprises shortly. It is more likely that a certain degree of collaboration between humans and cobots will take place where both sides will benefit from their strength [44]. The cobots are very good at repeating specific instructions for virtually unlimited time without any mistakes or tiredness (ignoring wear or program errors). Give the man a hammer and tell him to hammer fifteen nails into the wooden block, in a specified order, each with precisely four hits with a strength of 100 N. Even if he can use the right strength of the hit, after a few minutes, his hand starts to hurt, which will affect his strength; later, he will not remember the order of the nails or how many times he has to hit them and why he has to do such a pointless job. The robot, on the other hand, will manage to repeat this task precisely, without any complaints, as long as there are enough nails and wooden blocks in the world.

Just like the cobot, the human has its advantages – primarily, he can think (which can also be a disadvantage sometimes but let's not take this into account for now), so he often can handle unusual situations, whereas the cobot would fail – e.g., when some part is missing, when something slips from gripper, etc. Moreover, humans have very versatile end-effectors, also known as hands, which allow them to use various tools and effectively grasp objects of arbitrary shapes.

I see the potential of deploying cobots in the Small and Medium Enterprises (SME); therefore, I have selected cobot programming as the primary use case for *bringing the interaction between humans and computers to the task space*.

This chapter presents an overview of collaborative robots, end-user robot programming, AR and VR.

2.1 COLLABORATIVE ROBOTS

Cobots are among the immense hype of recent innovations in industrial automation. This chapter will summarize what the cobot is and what it is not. Several examples of cobots and their applications in the industry are presented.

Historically, industrial robots were used almost exclusively in big factories, with large batches, long-term production, and little or no need for retooling or changeover [44]. This is about to change with the rapidly emerging number of collaborative robots or cobots. Standard ISO 8373:2012 [65] defines a collaborative robot as a *robot designed for direct interaction with a human*. Cesta et al. [31] consider every robot able to work alongside the human without a safety fence as a collaborative robot. Nowadays, cobots are deployed not to the tasks where direct interaction with humans takes part (see Fig. 2.1); they are primarily used because of their implicit safety [63], which enables the building of robotic cells without fences or laser barriers. Such a work cell is more flexible, cheaper, and takes up less space in the factory. It is, however, expected that close collaboration of the cobots and humans will be the preferred variant for many tasks in the future [44].

As already mentioned, the main attribute of cobots is their safety. Nevertheless, with this safety, several drawbacks and trade-offs came to light. One of the most visible drawbacks is the rapidly reduced movement speed of the cobot. To not be physically able to hurt humans, weak motors are usually used in cobot joints, which results in their ability to manipulate with lower payloads than their industry relatives. Cobots are usually built ergonomically; they lack sharp edges, their joints are designed not to pinch a human's arm, and they often utilize a soft cover, which decreases the potential damage to the human body in case of contact. On the

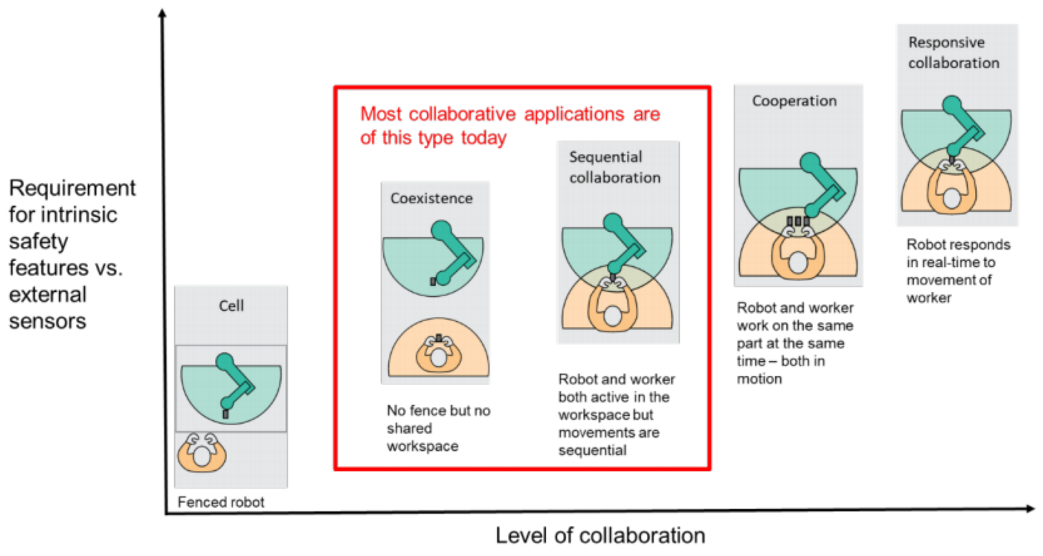

Requirement for intrinsic safety features vs. external sensors
Cell
Fenced robot
Most collaborative applications are of this type today
Coexistence
No fence but no shared workspace
Sequential collaboration
Robot and worker both active in the workspace but movements are sequential
Cooperation
Robot and worker work on the same part at the same time – both in motion
Responsive collaboration
Robot responds in real-time to movement of worker
Level of collaboration

Figure 2.1: Different types of interaction with robots. Reprint from [63].

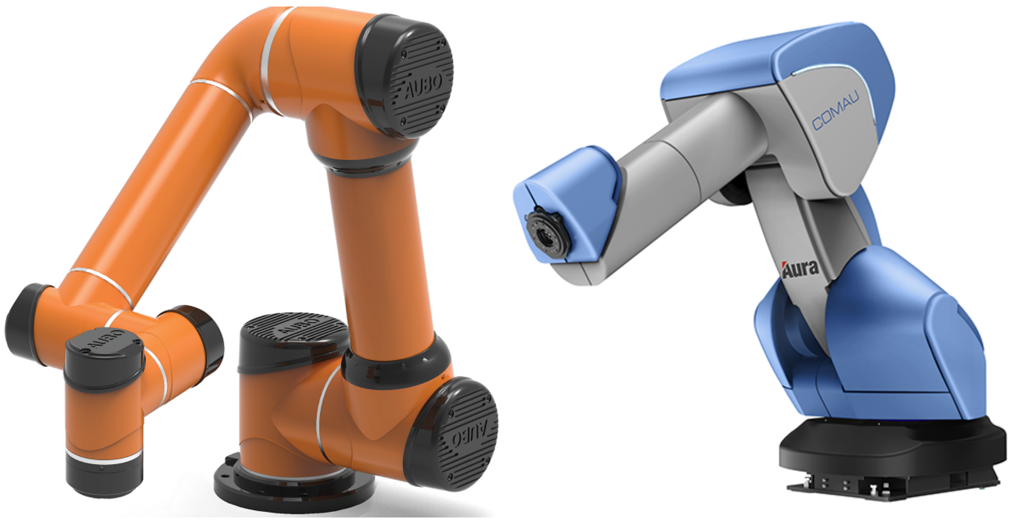
other hand, this protective cover is usually more vulnerable to damage in industry settings.

Various collaborative robots are commercially available, which differ in design, payload, reach, programming methods, and how they handle operator safety. ABB's YuMi (You&Me) represents a two-armed robot focused on assembly and face-to-face collaboration with the human operator. It is partly anthropomorphic as it consists of the torso with two arms but no head. The YuMi is inherently safe, which means it is too weak to hurt humans by accidentally crushing into him (robot body only, end-effector and manipulated object has to be considered separately) [114].

The most common way to achieve the robot's safety is so-called *joints sensing* [114]. It means that the robot is monitoring forces applied to his body through his joints, either by measuring the motor's power input or using force-torque sensors in the joints. This method is used, for example, in robots by Aubo (see Fig. 2.2a). By measuring forces, the robot can have bigger payloads and reach; for example, Aubo i5 is a 6-axis arm with a reach of 924 mm and a payload of up to 5 kg.

Some robots use many tactile sensors all across their body to constantly check if the robot is "touching" something. This technology is called *skin sensing* [114] and is used, for example, by the Comau Aura robot (see Fig. 2.2b). It is a costly technology, but it is considered safe and allows even higher payloads and reach than previous technologies (up to 170 kg and 2.8 m with the Comau Aura robot). Although the robot is safe for the human operator, it still can badly hurt humans when it hits them with more than 100 kg weighing load.

Comprehensive lists of currently available cobots can be found in the Collaborative robot buyer's guide by Robotiq [114] and an overview by Villani et al. [146].



(a) Aubo i5 robot. (b) Comau Aura robot

Figure 2.2: Examples of collaborative robots[1].

Although it looks like these small, slow, weak (except for the Coma Aura robot) cobots are useless in industry settings, the opposite is true. Unlike humans, the cobot does not need a smoking, toilet, lunch, or any other kind of break, except when it breaks itself. One cobot replaces three people during the three-shift operation, does not complain about weekend work, and has no hangover days. Because of that, the cobots might be faster overall, even though their cycle time could be lower than humans.

Applicability of cobots in SMEs

As was already mentioned, not every industry task is suitable for cobots, but there are many. A widely common task is assembly. The cobot could perform the task in coexistence with a human operator, using a sequential collaboration [127] or even real-time cooperation [24]. This task could benefit significantly from the combination of sensory and tactile abilities of humans and the precision, power, and repeat accuracy of the cobot [127]. Another prevalent task is pick&place [48], where the cobot is supposed to move workpieces from one place to another, put them inside machines or pick them from structured blisters or unstructured boxes.

The cobot could also work as a fixture [96], where the cobot holds a workpiece in a specific, ergonomically friendly position so that the human could operate on the workpiece. A fetch task is similar [84], where the robot picks up objects from

[1] Credit: https://roboticsbiz.com/top-collaborative-robots-in-the-market-with-pictures/ and https://www.comau.com/en/competencies/robotics-automation/collaborative-robotics/aura-collaborative-robot/

feeders or input boxes and hands them to the human, possibly based on the state of the assembly process.

Co-manipulation tasks [81] are possible where the cobot holds the weight of a giant or dangerous object, and the human leads the robot to a specific place or through a specific path. The nature of the cobots allows the operator to drag the robot and physically manipulate it through the environment [30].

Visual inspection is a common task [18, 89] as it allows the cobot to inspect the manufacturing products in a deterministic way. The cobot, holding a camera or other sensor (e.g., LiDAR or sonar), moves around the inspected object to check its quality or production accuracy. The cobots could also serve in surface finishing tasks, such as sanding [51], grinding [120] or painting [58]. Screwing [160] and drilling [4, 11] are other examples of cobot-friendly tasks.

2.2 END-USER ROBOT PROGRAMMING

Robot programming is an expensive task, as it requires a highly skilled programmer. Besides the common skills required for the programmer, such as knowledge of various data structures or algorithms, the robot programmer requires knowledge from other disciplines, such as control engineering, mechatronics, or computer vision [7]. Several end-user robot programming techniques emerged to lower the knowledge needed and the overall price for the programming process.

The end-user robot programming (also known as no-code programming) is a tiny bit of a broad area of end-user programming, in which overarching methods allow non-programmers to specify the behavior of some mechanical devices [7]. End-user programming is involved in many technologies of everyday use, such as animation, e-mail, gaming, home automation, or spreadsheets. The approach enables ordinary users to manipulate data structures and automate processes without extensive knowledge of low-level programming languages. The end-user programming is typically intended for personal usage, contrary to traditional programming [72].

Two main approaches are widely used in end-user robot programming, the *demonstration of skills* and *specification of programs* [7]. The former, also known as the Programming by Demonstration (PbD), utilizes human guidance to learn robot new skills [19]. It is inspired by how humans learn a new skill by imitating another human. The application varies from leading the robot through a specific path several times, where the robot records the joint states and the speed of movement and create a generalized trajectory; to observing the human performing a particular task using a camera or other sensors and extracting a generalized description of the needed actions, concerning the task objects in the environment [83].

Interfaces, utilizing PbD, provide an easy and natural interaction between humans and robots. Modern methods allow the robot to repeat the demonstrated tasks under varying conditions or in new scenarios because of the generalization of the task [146].

The *specification of program* approach seeks to simplify the manual definition of the structure and logic of the robotic program for the end-users [7]. It involves several phases, during which the user specifies the workplace, the program logic, authors the program for correctness, and last but not least, executes the program itself. Computer vision techniques are often utilized to specify task objects [62] or calibrate the workspace [97].

Different levels of robot's actions abstractions are involved, such as the lower-level definition of end-effector poses, a.k.a. waypoints [111, 158] or higher-level definition of tasks, objects and actions performed on them, such as picking, placing or surface finishing [62, 100] or object assembly commands [128].

Using visual programming, the user arranges the visual elements, which are automatically translated into executable artifacts [132]. Various examples of this approach exist in end-user robot programming, including using Behavior Trees [104], block-based visual programming languages [62, 94, 137] or flow diagrams [14]. Visual programming suffers in terms of visualization of spatial information; therefore, methods authoring this problem emerge [39, 60]. Much AR- and VR-based approaches have emerged lately, which will be more thoroughly investigated later in this thesis.

2.3 AUGMENTED AND VIRTUAL REALITY AND THEIR USAGE IN THE INDUSTRY

The VR immerses the user into a completely simulated virtual world [152]. It utilizes the Head-Mounted Display (HMD) in combination with the head and possibly eye tracking to provide the user the feeling of presence in the virtual world. Walsh et al. [148] state that VR must fulfill the three properties: (tele-)presence, interactivity, and immersion. In other words, the users must feel that they are physically elsewhere and can manipulate the virtual world in real-time. "*VR leverages immersive technologies to simulate interactive virtual environments or worlds where users become subjectively involved and feel physically present*" [148].

In the industry, the VR found an application in the virtual training of new employees. Traditionally, the training for the new employee is provided by an experienced colleague, which is costly because it prevents the trainer from his usual work (unless they are a full-time trainer). Bellaouna [16] proposed virtual training for handling a special vehicle using an immersive VR application. The advantage of such an approach is that the new employee could train at home. Several other approaches were proposed for machine tending [99] or assembly [3, 66, 159].

The VR training system could also be used for preserving experienced user knowledge [99, 117]. The expert worker performs assemblies using their experience first; later, the process mining algorithms obtain the assembly models and provide them to the trainee workers to improve the process [117].

Interesting usage of VR is remote monitoring of an actual state of production [161], using a digital twin of the actual device and observing it using an immersive HMD. The digital twin, in combination with the VR environment, could also be used to assess ergonomics and risks in a robotic collaborative workplace [55] before the actual deployment of such a workplace. In the chemistry industry, VR could help in robotic workplace prototyping, training, or lowering the risk of injury [47]. Žídek et al. [161] propose a VR-enabled virtual assembly tool.

The VR also found its usage in robot programming. Burghardt et al. proposed a robot programming method using VR and digital twins [29]. It enables the robot to imitate the movements performed by an experienced human worker in VR. Holoubek et al. [57] have shown that the combination of offline robot programming using the PC desktop software and immersive VR environment shorts the time needed for industrial robot programming. The robotic cell and corresponding program created in ABB's Robot Studio is exported into the VR environment. The user can visualize it there and manipulate specific parts of the program, such as spatial parameters of move instructions, speed of movements, and others. Manou et al. [88] proposed another VR-based method for robot trajectory programming using a hand-held teaching tool, similar to the actual end-effector tool, suitable for tasks such as welding, deburring, or cutting.

Contrary to the VR applications, which use almost exclusively the fully-immersive HMDs, the AR application preserves the ability the see the real world while providing the interaction between it, the user, and the digital content, superimposed over the real world [155]. The AR could utilize many different devices, varying in cost, possible usage, interaction modes, and ergonomic properties.

One of the cheapest AR-enabled devices is a mobile device, such as a cell phone or tablet [27, 53]. The disadvantage of these devices is the necessity to hold the device in one or both hands, which limits the possible applications. The huge advantage of such a device is its widespread among people; therefore, the people are familiar with the device's controls using a touch screen. The device's screen shows the camera's output pointing toward the human's gaze. Digital content is superimposed over the camera image and spatially correlated to the real world, using tracking and 3D registration of the actual scene [75]. The user may interact with the scene by physically moving the device in the environment and combining on-screen control and 3D widgets.

Alternatively, an HMD device could be used, such as Microsoft HoloLens [38, 77, 145], which frees the user's hands. The HMD intended for the AR are so-called see-through, meaning the user can see the real world while wearing the HMD. Because of a richer set of sensors compared to standard mobile devices, the HMD typically offers more precise localization in the world and stable presentation of the digital content [86]. On the other hand, wearing an HMD for a more extended period could be tedious. Since the HMD does not utilize the human's hands, they could be used for interaction with the digital content, which could be natural for the user [86]. The voice is also a standard interaction modality for the HMD devices, both AR and VR. Although controlling the interface by hand gestures or voice could be natural for the users, it could be challenging to properly recognize the human's orders, especially in an industrial environment, due to loud noises and protective gloves.

Another alternative is the usage of projection [87]. The SAR uses the projected light to change the appearance of the physical environment [17]. The advantage of this approach is that the user does not have to hold or wear any potentially heavy and cumbersome device [126]. Moreover, the superimposed digital content is visible to more people simultaneously. The disadvantage is the ability to project only on surfaces; therefore, expressing 3D information in the free space is challenging without an additional medium. Like the HMD, gestural and voice interaction are common modalities for SAR; therefore, they suffer from the same disadvantages in the industry sector. Moreover, contact sensing could be utilized [34]. Additional physical devices could be used to control the SAR, such as smartphones or tablets [103].

The AR is involved in various tasks in the industry, such as work cell design [73, 103], assembly guidance [5, 20] or robot programming [111, 158].

During the manual work, the workers usually have printed materials with instructions, varying from simplified graphics information to complex manuals. Alternatively, a digital display is available next to the workplace with an application containing the same information as the printed materials, extended by videos. A common disadvantage of these solutions is the information displayed out of context of the actual manufacturing process, so the worker needs to constantly switch their attention between the workpiece and the manual and make a mental transformation between the 2D information on the paper or display and the real, 3D workpiece. The AR helps to solve this problem by providing the same information in the context of the manufacturing process [5]. Besides the visualization of the manufacturing process, the process monitoring with a warning system could be incorporated into the AR interface [23].

An AR application often provides assembly assistance [1, 20]. Deshpande et al. proposed AR application using HMD as an aid to support Ready-to-Assemble furniture [38]. They claim that such an interface helps users improve spatial problem-solving abilities. Their user study showed a significant reduction in the time needed for the assembly using AR-based support.

3 HUMAN-ROBOT INTERACTION AND PROGRAMMING

Novel interaction methods with robots must be developed to enable SMEs to incorporate robotic solutions in their productions. Reasons for such a need are mainly small production batches and the associated frequent changes in production or the high price of highly skilled robotic programmers.

Most robot producers develop their robot programming software, which is often compatible only with their robots. This software usually involves proprietary language integrated into complex online or offline programming systems, which allows simulations, verification, or synchronizations of multiple robots. Many robots could be programmed using traditional universal programming languages like C++, Python, and others. Robotic frameworks often came into play to help the operator with common problems, such as hardware abstraction and communication. An example of such a framework is, e.g., ROS - Robot Operating System [110].

When a robot is supposed to operate in a non-structured environment, it needs to be aware of objects and obstacles in such an environment. At the same time, the human operator needs to know whether or not the robot understands the environment correctly. To address these issues, we proposed a collaborative workspace in a paper [93] using projected SAR, which enables the user to observe how the robotic system understands the adjacent world and simultaneously provides the user with a simplified method of robotic programming.

As the robotic tasks are often spatially related to some specific place in the environment, e.g., inserting a workpiece inside a pressing machine, it would be beneficial to express this relation in the real space, using the SAR. When developing our methods, we were inspired by a typical workbench used for manufacturing in various factories worldwide.

For a user to be able to communicate with the robot, some input and output modalities have to be incorporated. There are different modalities for transmitting information from humans to robots and vice versa. A typical example of the former is speech - the human tells instructions, and the robot executes them. An example of the latter is text printed on some output device (e.g., screen), which humans read. Different modalities are suitable for different interactions and transmitted information, such as in the speed of transmission, reliability of information delivery, and others. It is common to use multi-modal systems where several modalities apply either for transmission of the same type of information (i.e., redundant modality) or different types of information (complementary modality). An example of a multi-modal system is an ordinary computer, where the user uses a keyboard for text input and a mouse for 2D navigation on the screen.

The following chapter presents the preliminary research, which led me to state the goal of the thesis.

3.1 INTERACTION NEEDS FOR COBOT PROGRAMMING TASKS

During the years, we have visited many factories of our industrial partners to see various robotic solutions. Based on these experiences and state-of-the-art research, we have defined a set of typical robotic problems and basic interaction needs for programming them:

1) Selecting the object of interest: The robots typically work with real-world objects. They could be the workpieces the robot manipulates or machines the robot controls. Suppose the programming method is aware of the objects, either by sensing or manually defined by the programmer. In that case, the robot programmer could use the real-world objects to define actions for the robot, e.g., put the object into *this* box. Therefore, selecting the object of interest is one of the most crucial interaction needs for the robot programmer.

2) Selecting the specific point in the environment: The robot programmer often must specify a point in the environment, either in the free space or on the real object. An example is spot welding, where the programmer is setting the points for the robot's end-effector on the workpiece or picking objects using a suction cup, where the programmer sets the picking point. A path for the robot's end-effector could be defined by combining several consecutive waypoints.

3) The specific robot's end-effector pose definition: It is crucial to specify an exact robot's end-effector pose for various assembly or insertion tasks, where the orientation or direction of the end-effector has to be specified. Similarly, the gripping pose has to be specified when picking an object using a gripper.

4) Assembly constraints definition: The programmer has to define the ordering of steps during an assembly or the spatial parameters of the assembly, e.g., the orientation of different parts making a product.

3.2 COBOT PROGRAMMING SCENARIOS

We have defined two testing table-top scenarios using a subset of presented interaction needs. One incorporates the collaboration of humans and robots; in the other, the robot works independently. The first one, initially presented in the paper [92], deals with assembling a simple stool composed of several wooden blocks (see Fig. 3.1). In the scenario, these blocks are stacked in a gravity feeder on the side of the table, and the robot serves as a companion to the human. The robot handles the blocks from the feeder to the table while the human is responsible for the assembly process. Besides, the robot has to put glue inside holes on the wooden blocks. This scenario incorporates formerly presented needs 1 (select wooden block to be taken), 2 (specify a place on the table where the object should be placed and specify a hole on the object where glue should be applied), and 3 (define a gripping pose for a wooden block in gravity feeder).

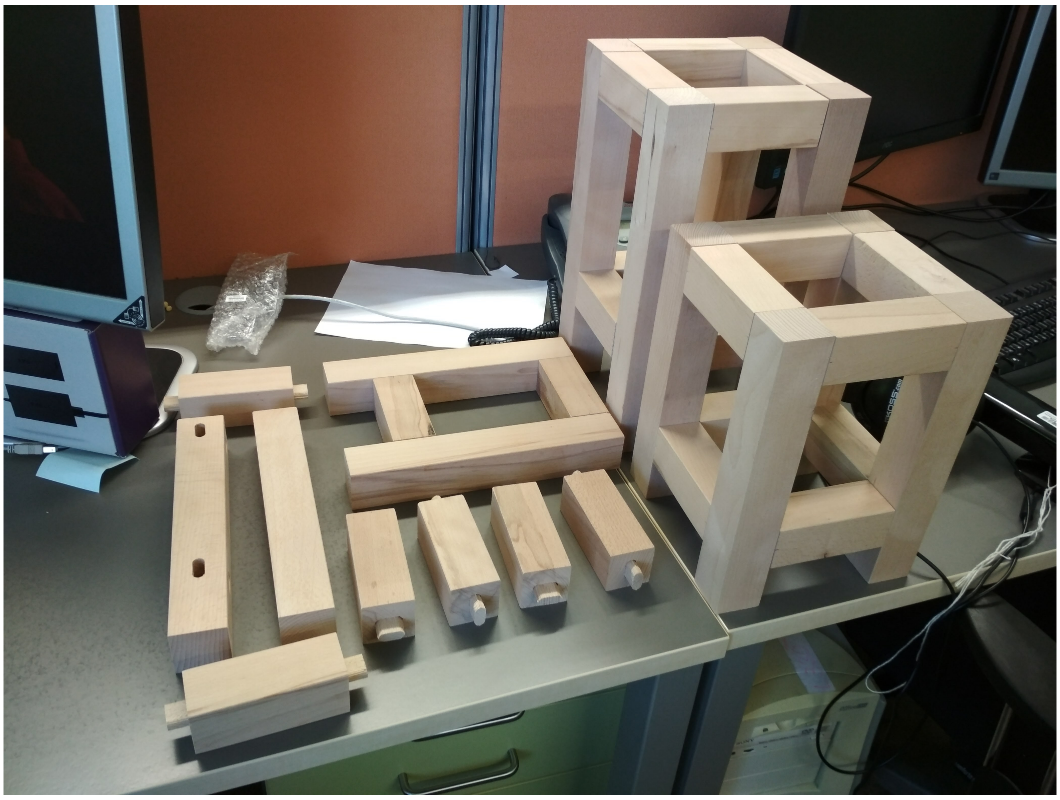


Figure 3.1: The stool used for scenario 1. On the right side, two stools of different heights are assembled, and on the left side, the individual parts of the stool are placed.

Initially proposed in the paper [67], the second scenario deals with offline Printed Circuit Board (PCB) testing in an SME company. In this company, relatively small batches of PCBs are tested, and items' storage is highly variable. Therefore, a program must be adapted approximately once a week. In this case, the robot has to pick either unorganized items from crates or organized ones from blisters, place them inside the testing device, run the test and, based on the result of the test, print a corresponding label, put it on the object, and place the object on one of two boxes, again, based on the result of the test. So far, the process was done by human operators, but their time was not used efficiently, as they are idle for up to several minutes while the test runs.

Moreover, the work is highly stereotypical. The goal was to optimize the use of a qualified workforce as most operators could be reassigned to more creative work after robotizing the process. The rest could be trained to be able to adapt programs of testing workplaces when needed and to supervise multiple workplaces during execution. This scenario uses interaction needs 1 (selection of the object to be picked from the table), 2 (place inside the box, where PCB should be placed), 3 (exact position of the object inside the testing device), and 4 (how to put a label on PCB).

3.3 INTERACTION MODALITIES END ERROR EFFECT

Different interaction methods were investigated to enable fluent interaction between humans and robots in simplified robot programming, service robots, or mobile platforms. To design and create a prototype of a robotic work cell usable to verify use cases from the previous chapter, a set of input modalities for control and programming the robotic arm had to be defined. We have selected several modalities from everyday communication between humans and existing machines and some not-so-common modalities. The first selected modality was *hand gestures*. Every human uses gestures daily, e.g., for pointing, confirmation, rejection, or simply saying hello. Therefore, gestures are a relatively universal method of interaction. As precise detection of bare hands in free space could be problematic, we also decided to utilize a 6 DoF tracked device, mainly for pointing tasks, which the user holds in their hand.

As most people nowadays use some touchscreen devices (e.g., cell phones or tablets), it was natural to use some touch-sensing modalities. One of them is a touch-enabled screen placed next to the working bench. This modality partially violates the requirement for interaction directly inside the robot's working space. However, the popularity of different touch-enabled devices convinced us to use them for usability evaluation. Another touch-enabled device was the working bench covered by multi-touch foil. In combination with the projector above the table, a large touchscreen is emulated.

The last selected modality was direct manipulation with a robotic arm. This modality could provide high precision for specific tasks (such as loading workpiece inside the machine, welding, or polishing). Moreover, it is already a widely used modality for robotic programming.

A user study examining the usability of selected interaction methods under different error rates is presented in the paper [93], involving 39 participants. Selected tasks were inspired by typical industrial robotic tasks: assembly, pick&place, and welding of points and seams. The study was conducted in the form of Wizard-of-Oz (WoZ) to control the rate of errors for each input modality and eliminate implementation issues. The human observer in another room was evaluating the

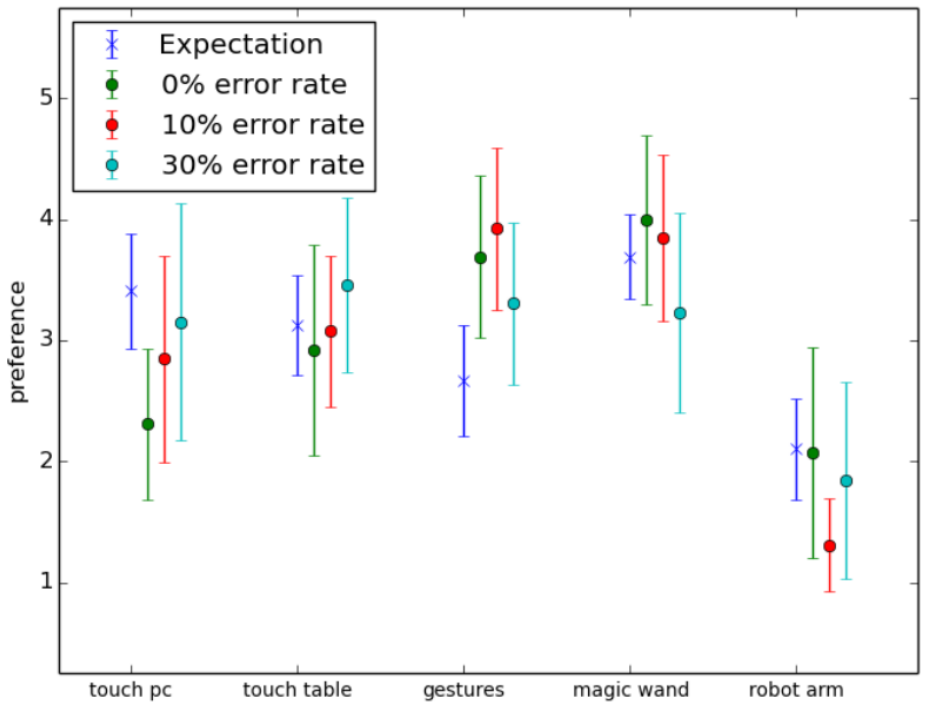


Figure 3.2: The participant's ranking of different modalities for *select point* task and different amounts of intended interaction errors, accompanied by the initial expectation of the users. The results for other tasks were similar.

user's input, using video and audio stream, and was providing proper feedback to the user with a projector and computer screen.

The Fig. 3.2 shows a graph of participants' ranking of different interaction modalities. Each participant tried all modalities and was told to order them according to their preferences. Besides, prior to the experiment tasks, each participant had to order the modalities according to their expectation. Each participant was assigned a specific amount of interaction error, inserted intentionally into their interaction using the WoZ approach.

Regardless of the number of interaction errors, gestures and the 6 DoF device were among the most preferred modalities for all interaction tasks. As these two modalities were almost identical in this experiment, except that with the 6 DoF device modality, the user had to hold the device in his hand, we initially decided to choose gestures as one of the main input modalities for future research. After the additional test with the available gesture detection, we were forced to reconsider this approach because of the slowness and inaccuracy of pointing and confirmation gestures. Therefore, the gestural control was not reflected in my following research. Right after the gesture controls, the touch-enabled desk and computer were ranked.

In scenario 1, most interaction happens directly on the table, with little need to specify 3D spatial parameters. The scenario does not use any additional machines;

therefore, the user interface could be projected on the table. We have selected the SAR interface in combination with the touch-enabled surface of the table. For setting the 3D spatial parameters, i.e., the object's position in the gravity feeder and locations for glue application, we have selected the direct manipulation with the robotic arm.

For scenario 2, the AR on a mobile device with a touchscreen was selected as the primary user interface. The interaction modality is a touch control on the device's screen. This scenario assumes the presence of additional objects on the table, such as the tester device and several boxes for the PCBs, and the need for specification of several 3D spatial parameters such as the PCB's inserting position to the tester device. Therefore, the mobile AR seems more appropriate in this scenario.

4 THESIS STATEMENT, GOAL, AND RESEARCH OBJECTIVES

The thesis statement leading my overall research, described in this thesis, is formulated as follows: ***Bringing interaction between humans and computers to the task space may enable non-expert users to create and adapt robotic programs.***

In this context, the non-expert user is defined as a person with no or minimal experience with robot programming but usually has moderate specific domain knowledge, e.g., knowledge of particular manufacturing processes. Providing a simple and usable robot programming method for those people could increase the automation share in SMEs, dramatically decreasing the price for robotic work cell programming.

For this thesis, the robot means a collaborative industrial robot working close to a human. Although, in theory, the proposed methods should be universal and work with any robot or mechanical device, the primary targets are collaborative robots.

The main goal of the thesis is to develop an efficient method for end-user cobot programming in AR, which allows a regular shop-floor worker to (1) create a robotic program, (2) comprehend existing programs, (3) adapt such a program to new conditions, and (4) collaborate with the robot on the performed task. The proposed goal forms four research objectives:

O_1 **– Define visualization and interaction methods suitable for such interfaces**

The programming system has to visualize various information to the user, such as the current system state or the robot's intentions. Almost every user interface needs an appropriate interaction method as well, used for inputting data and orders. As was already stated, interaction and visualization should somehow occur in the task space, so the second objective deals with searching for such methods and evaluating their acceptance by the users.

O_2 **– Design a method for defining program flow**

Once the user can see the information from the system and has a method of how to input new information or alter the existing one, they can start the programming process itself. An appropriate method for defining the program flow has to be designed to do so. This method should allow the user to define an order of individual actions and allow them to define standard programming features, such as conditions or cycles.

A crucial part is the understandability of the program's visual representation for the users. Understanding a program's purpose, seen for the first time (or after some time), is vital for a fast adaptation of the program to new conditions.

O3 – **Design a method for defining and visualizing the program's spatial parameters**

As was already mentioned, every robotic program's essential part is manipulating the real world and its objects. The programmer has to specify several spatial parameters when defining a robotic program. At the same time, the programmer needs to observe the previously created parameters to understand the program or to alter the parameters. A method for easy definition and clear visualization of such parameters is crucial for the end-user robot programming tool to be successful.

O4 – **Evaluate the proposed method with non-programmers, concerning usability, mental workload, and user experience**

The thesis's final and inevitable objective is to evaluate all proposed methods with real users, focusing on non-programmers, i.e., potential future users of tools developed based on the proposed methods. The evaluation should focus on the functionality and usability of the methods and involved user interfaces, perceived users' workload, and user experience.

Contributions

The main contribution is a novelty method for simplified robot programming, using so-called Spatially Anchored Actions (SAA), based on the definition and visualization of the program's spatial data in the task space, i.e., in the real world, where the performed task takes place. Besides, a fully functional prototype using this newly proposed method has been developed by me. More than 70 participants were involved in user studies throughout the work on my thesis. The following chapters describe all my contributions in detail, starting with projected SAR for programming the table-top scenarios, which later transformed into initial 3D prototypes using a mobile AR, to propose the method mentioned earlier and evaluate it using a functional prototype.

Part II

PROJECTED AR AND TABLE-TOP ROBOT PROGRAMMING

To bring the interaction into the task space, the world had to be utilized and enriched with an interactive user interface. It must provide the user with information about the controlled system and, simultaneously, enable them to control the system naturally. One way to do so is the utilization of so-called Spatial Augmented Reality, using the combination of the projected user interface and the touch-enabled surface of the workbench. The user can see the system state within their workspace and control the projected elements using well-known touch control. The interface provides the user with a simple and understandable way to input and visualize the spatial parameters according to the stated objectives. The following chapters present the proposed method of spatial programming using a projected user interface, which was evaluated with 9 participants across two qualitative user studies. It was shown that our proposed method performs better than the standard robot programming method in terms of programming time, usability, and user experience.

5 INTERACTIVE SPATIAL AUGMENTED REALITY IN COLLABORATIVE ROBOT PROGRAMMING

The previously described scenario 1 (Section 3.2) deals with human-robot collaboration on assembling a wooden stool. In this scenario, the robot acts as an assistant to the human. The robot hands different stool parts to the human, who assembles the resulting product. The human operates in this scenario as both programmer and worker. Firstly, they need to teach the robot to correctly hands them the right parts at the right time; later, they collaborate on the assembly with the robot.

We needed to prepare a suitable workplace and a method to visualize the program to the programmer and enable him to set the program's spatial parameters. The intended programmer is non-expert; therefore, the interface must utilize a high level of abstraction, require no programming knowledge and visualize essential parameters only. Since the robot and the human share the workplace, the system must adapt to changing conditions, such as objects unintentionally moved by the human.

Following the stated objectives, we have proposed an interactive spatial user interface for simplified collaborative robot programming, using the projected SAR, so the programmer sees all necessary information projected in the task context on the workbench (O_1). The program's flow is projected as instruction blocks with the information of transitions between the instructions (O_2). Besides, the interface presents all detected objects on the table to support the programmer's awareness. Instead of teaching the robot the precise movements, the programmer creates a highly abstract description of the program, such as *pick any blue cube from this part of the table and put it on this place on the table*. The precise positions of the *blue boxes* are inferred during the execution phase; therefore, the system can adapt to changing conditions. To comply with the O_3, individual instruction's spatial parameters are projected on the workbench as manipulable widgets. For example, to set a position on the workbench to place an object, the programmer drags the projected outline to the desired position and sets the orientation of the outline using touch gestures.

A high level of abstraction was selected to simplify the programming process; therefore, no robot-specific knowledge is necessary to use the interface. The proposed method enables the user to set the task and execute it in close cooperation with the robot. The user interface provides the human operator with necessary information during the programming and execution phases.

A functional proof-of-concept system (see Fig. 5.1) was developed and evaluated with six shop-floor workers in terms of usability and mental or physical demands in accordance with the O4. The study aimed to validate the initial concept and to show the potential of SAR in similar scenarios. This chapter is based on our published paper [91].

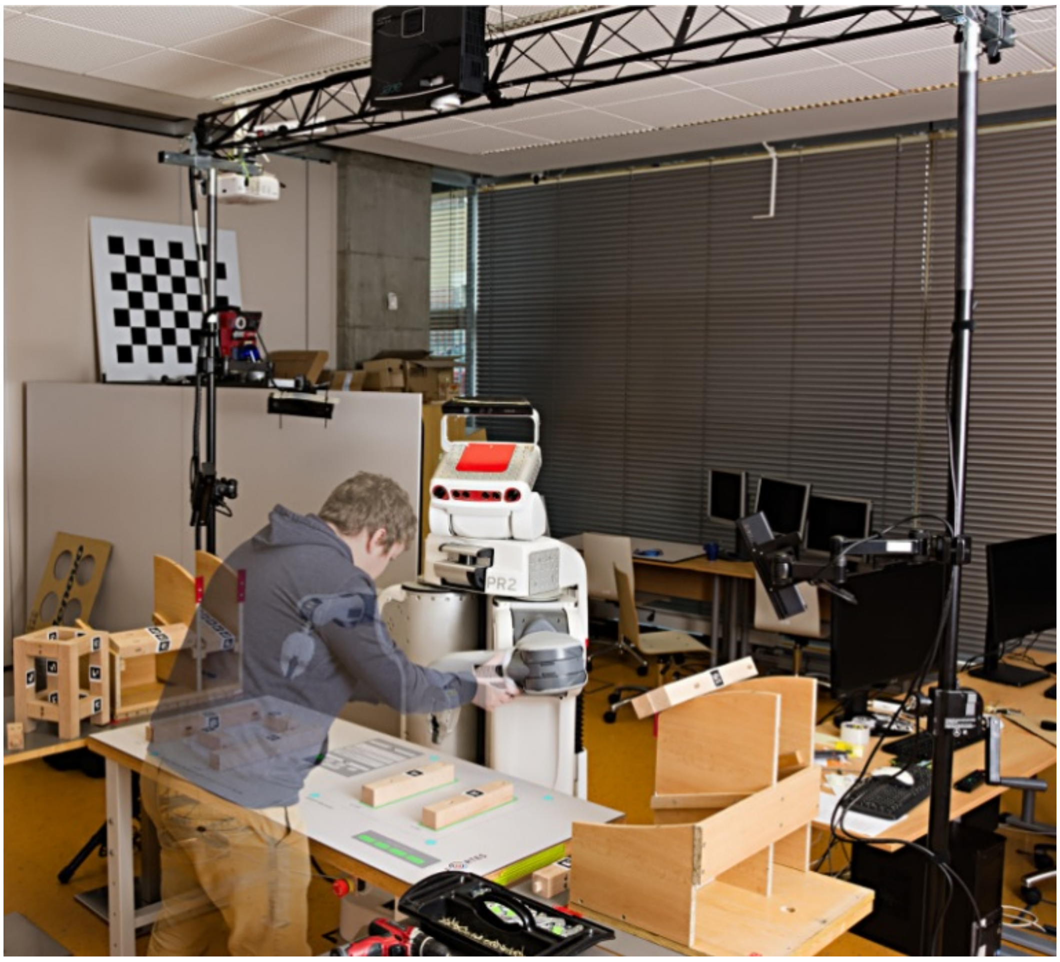


Figure 5.1: Setup of the novel interactive system concept where all the interaction elements (visualization and control) are gathered in a shared workspace (example of setting program parameters using a robotic arm and gestures; image edited).

5.1 RELATED WORK

Various approaches exist aimed at simplifying robot programming or supporting human-robot collaboration on a joint task. One of the techniques used to make programming robots more suitable for non-expert users is PbD. For instance, the approach proposed by Orendt et al. [101] was rated by non-expert users as highly intuitive. However, the tasks are pretty simple, and there is no feedback for the user. The approach uses so-called One-Shot PbD, where the operator leads the robot over the desired task using kinesthetic teaching (see Fig. 5.2). By utilizing the Online Oriented Particles Simulation approach [50], the system can generalize the single demonstration and adapt to new conditions, such as different robot starting positions or adjusted object positions. Their user study revealed the proposed solution's high intuitiveness and low perceived effort.

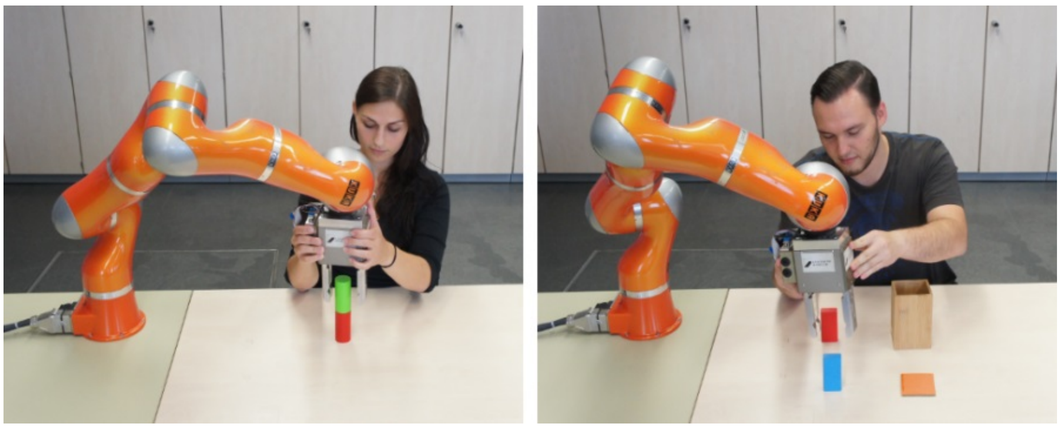


Figure 5.2: User programming the lightweight robot to solve a stacking (left) and sorting scenario (right). Reprint from [101].

Stenmark et al. [136] use kinesthetic teaching in conjunction with iconic-based programming (see Fig. 5.3) to enable users to create and edit non-trivial programs. While using a graphical user interface (GUI) on a standard monitor adds more control over the program and provides feedback, it also leads to attention switches. Based on their evaluation, the integration of rapid instruction, test, and execution cycle reduced the time needed by the expert by 80%, compared to traditional robot programming methods.

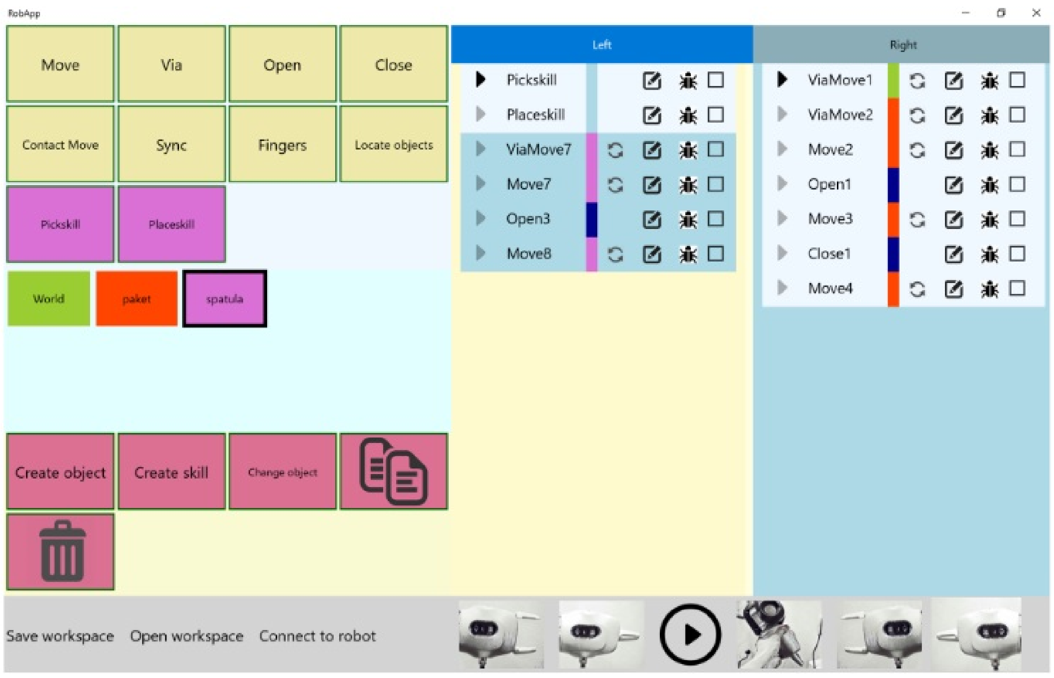


Figure 5.3: The graphical user interface. Reprint from [136].

The system described by Guerin et al. [52] uses behavior trees to represent the program and is successfully deployed at an SME. The program itself is created on the computer screen. The program's parameters could be set using GUI, object recognition, or kinesthetic teaching. The usage of behavior trees leads to high flexibility and the creation of reusable pieces of programs; however, it also inevitably leads to a more complicated GUI. Similarly, the system described by Huang et al. [62] enables users to create complex programs using kinesthetic teaching and object recognition. However, three different GUIs and voice input are involved. Moreover, its target user group consists of general programmers.

The previous approaches share a common disadvantage: The inability to show information within a task context. On the other hand, Sefidgar et al. [130] proposed the usage of physical blocks (see Fig. 5.4) to create a program that is highly intuitive as it requires no training. However, it is limited to trivial tasks. This so-called situated tangible robot programming enables operators to annotate objects, locations, or regions and specify actions to be performed and their order. They have conducted several user studies to evaluate the intuitiveness and learnability of the proposed approach. They stated that people could interpret, generalize and create various programs using the situated tangible programming.

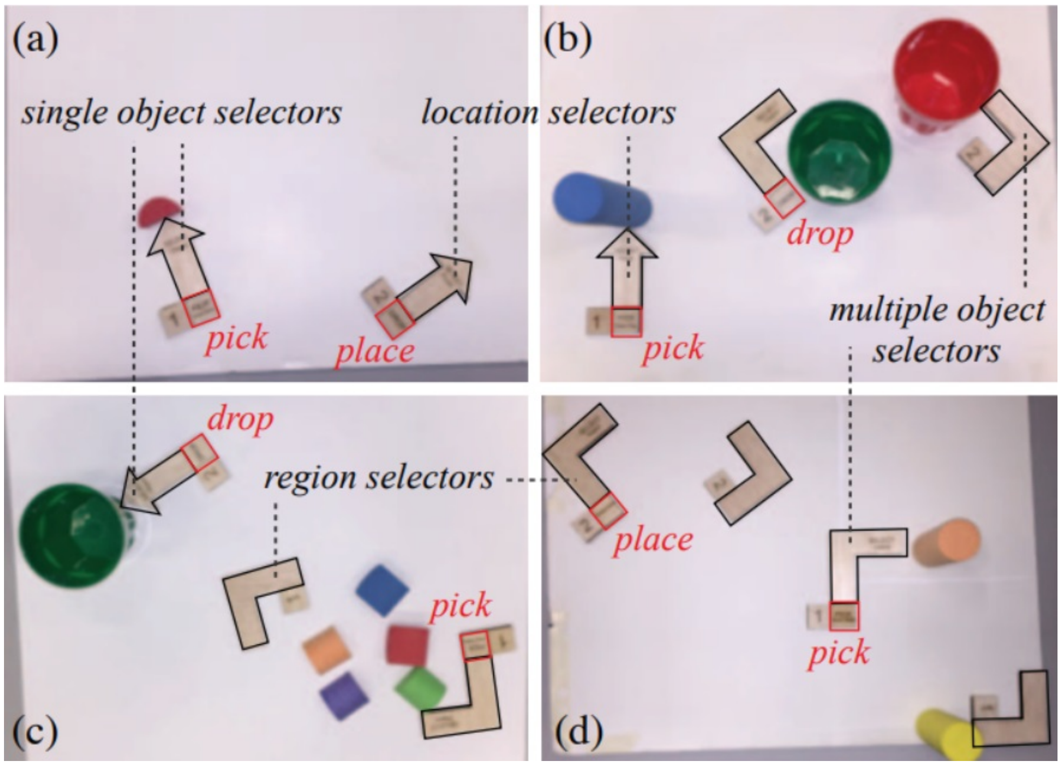


Figure 5.4: Example situated tangible programs in different scenes. Programs vary in the type of selection blocks used as part of the pick and place/drop instructions. Reprint from [130].

Recently, AR has been used to show important information within a task context. Probably the most common approach is to use a handheld device. Stadler et

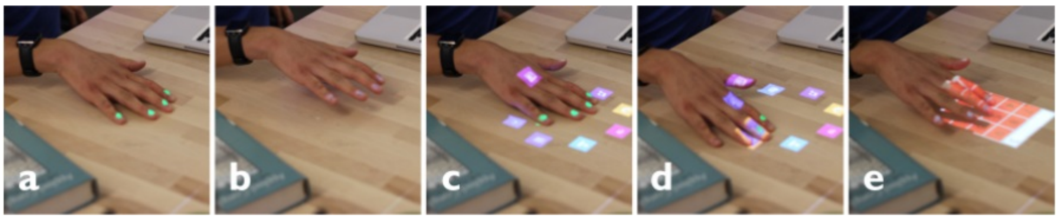


Figure 5.5: Summoning an application. a-c: The user taps twice with four fingers to bring up a launcher. d: The user moves to select the desired application. e: After lifting the fingers, the application is created. Reprint from [154].

al. [134] recruited robot programmers and evaluated a tablet-based AR interface for programming abstracted industrial tasks. The results suggest that using an AR may lead to a decreased workload and higher motivation to perform accurately. However, the usage of a tablet prevents the usage of both hands. A HMD frees the user's hands, and according to Rosen et al. [119] might lead to faster task completion and higher accuracy. Unfortunately, the currently available devices have a limited field of view. Also, a HMD probably would not be suitable for long-time usage.

On the other hand, SAR can show information in context, does not require any handheld devices, is suitable for long-term usage, and is visible to anyone. It was recently used to implement an interactive work desk [154] (see Fig. 5.5), using a projector-camera system, supporting several interaction behaviors, combining physical and virtual desk elements. The user controls the system using hand gestures.

Funk [45] proposed a solution (see Fig. 5.6), which utilizes the in-situ AR projections for displaying instructions to workers in the assembly cell scenario. Using predefined triggers, which react to existing boxes, assembly zones, or object detection zones, the system is able to lead the operator through the assembly process interactively.

For close human-robot cooperation, an operator's knowledge of data from the robotic system is crucial. Leutert et al. [78] designed a SAR system, able to show such robotic data and learn trajectories. This enables the operator to, for instance, know where the robot will perform its actions or which path it will follow.

5.2 PROPOSED APPROACH

We have proposed a novel approach for simplified robot programming, combining an interactive SAR, kinesthetic teaching, and object detection (see Fig. 5.7). The proposed approach meets the following requirements:

- Avoid any external control devices to interact with the system. Instead, adapt the shared workspace to be interactive to avoid switching the user's attention.

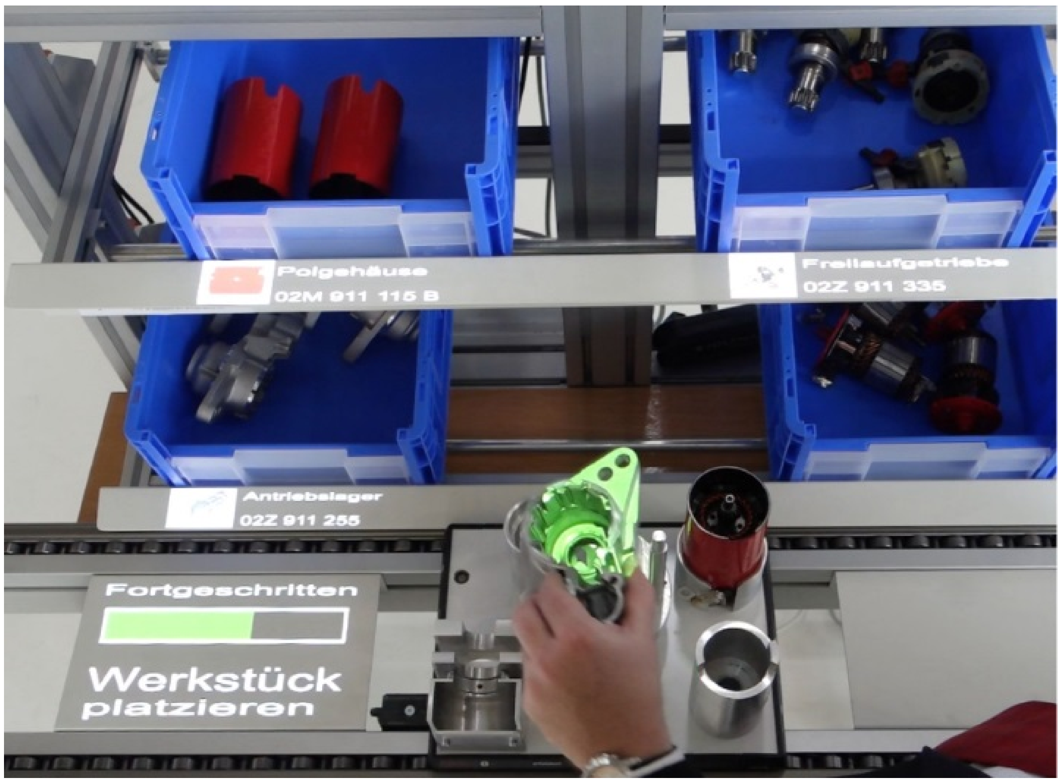


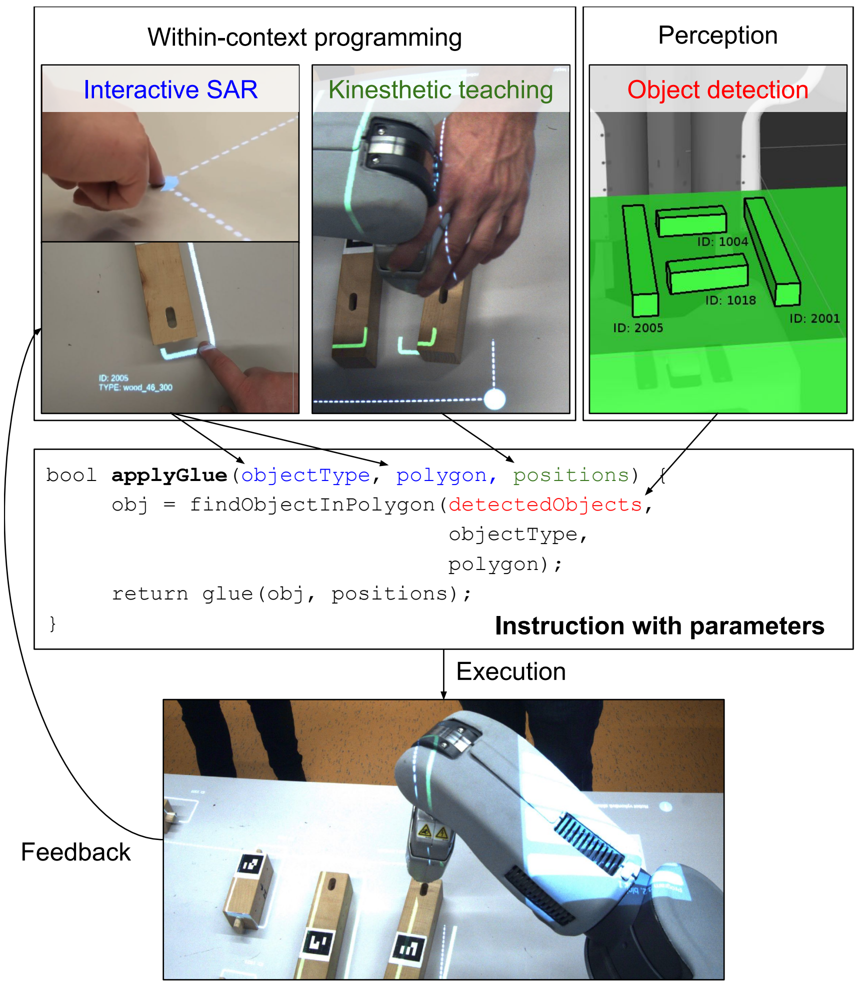
Figure 5.6: The projected feedback can be enhanced with additional information. At the beginner level, a video is shown additionally to text, image, and contour feedback. At the advanced level, only the contour is shown. At the expert level, no visual feedback is shown at all. Reprint from [45].

- Allow the non-expert users to work with the system. Utilize a high-level of robot program abstraction.
- Present the relevant program information, such as program steps or detected objects in the relevant context, to lower the operators' mental demands.

SAR enables us to fulfill these requirements in an intuitive and usable way. S touch-enabled surface was utilized to interact with the system, as it was ranked high in our previous research [93]. The robots' arms could be utilized to obtain required 3D spatial data, such as the position of objects in gravitational feeders.

As is typical for the SAR, the interface has to be minimalistic because it shares the same space as the other physical tools on the table, such as assembly parts or tools. The cognitive load could also be lowered by presenting only relevant information, depending on the current state of the system and the performed task [153]. The interface should clearly indicate the system's state and present an explicit representation of the robots' program in the environment's context.

The proposed approach works with high-level instructions with a high amount of underlying autonomy, which could lower the expressivity of the program. However, it significantly simplifies the programming process for the user. Using semantic knowledge of the environment in combination with visual perception, the



```
bool applyGlue(objectType, polygon, positions) {
     obj = findObjectInPolygon(detectedObjects,
                               objectType,
                               polygon);
     return glue(obj, positions);
}
```

Figure 5.7: Illustration of program parameters' definition (combination of manually set parameters by the user with perceived information by the system) and its execution with visual feedback.

definition of object picking instruction only needs two parameters: the type of object to be picked and the approximate position of the object. The user does not have to set a pre-picking and picking pose or specify when the gripper should close to pick the object.

5.3 PROOF OF CONCEPT SYSTEM

A proof of concept system has been developed to evaluate the proposed approach. The system allows end-user programming of selected industrial tasks.

Setup

The experimental setup (see Fig. 5.1) was designed to be easy to deploy and modular. It is centered around a standard workshop table equipped with a capacitive touch foil with two speaker stands placed on the sides, connected by a truss. The truss is equipped with an Acer P6600 projector. There is a Microsoft Kinect V2 camera on each stand for object detection and calibration of the system. On one stand, there is an additional Kinect for user tracking.

Each stand has a processing unit (Intel NUC) connected to the projector and sensors. The unit is connected to the central computer using a wired network. The system is designed to be modular in a way so that it supports $1..n$ projector units to overcome shadows cast by physical objects on the table (in the study, only one projector was utilized). The system contains calibration features for mutual calibration of cameras, projectors, robots, and touch-enabled surface.

As a demonstrator of a near-future collaborative robot, we use the intrinsically safe robot PR2. The robot provides an additional set of sensors (Kinect and cameras on the head, cameras in the forearms), which are also used for object detection. In case of an emergency, there is also a physical stop button under the table which shuts down the robot's motors.

The system is based on the ROS framework [110], a set of software libraries and tools intended to simplify the process of building complex robotic applications and systems.

System design

The system's state and behavior are defined and controlled by the central node, and an arbitrary number of user interfaces can manipulate it. Two interfaces were incorporated in the prototype used for the study:

- Graphical user interface, projected on the touch-enabled table.
- Sound interface, providing audio feedback for confirmation of action or errors.

As stated above, all system parts must be mutually calibrated first. Calibration of the Kinects utilizes an AR tracking library[1] to detect three markers placed on the table. One marker serves as the origin of the coordination system, while the two others determine the X and Y axes. The PR2 robot is calibrated similarly, using a head-mounted Kinect. As the robot's Kinect is too close to the table to perceive all the markers at once, the calibration procedure moves the robot's head to observe them one after another.

Each projector displays a checkerboard pattern to calibrate the projectors, and its corners are detected using already calibrated Kinect cameras. In order to calibrate the touch-enabled surface, several points are projected on the table, and the

[1] `http://wiki.ros.org/ar_track_alvar`

user has to click them. Then, a homography is computed and used to convert the internal coordinates of the touch device into the common coordinate system.

To avoid the object detection imperfection for our presented study, each object used in our study has a set of two AR tags printed on the body, and multimarker detection is used to gain a unique ID of the object and its pose. Each object has an object type and a bounding box defined. The manipulation pipeline is based on MoveIt! [35] and a library for grasp planning[2]. The marker-based detection would be replaced by a reliable industrial-grade object detector in real application.

Program representation

The program in our system is a set of instructions collected into program blocks. Each program contains $1..n$ blocks; each block contains $1..n$ instructions. Every instruction execution can result in success (e.g., a successfully picked up object) or failure (e.g., failure to apply glue). Based on the result, the next instruction is determined. This approach makes simple branching and cycling of the program possible (e.g., picking up objects from a feeder until the picking up fails, i.e., until there are no objects left). For an example of a program structure in the form of a graph, see Fig. 5.10.

Contrary to the conventional methods of programming robots, no precomputed joint configurations or arm paths are stored. By combining the perception capabilities of the system and on-the-fly motion planning, we do not rely on, e.g., storing exact object positions.

It can be expected that the program's parameters will be changed more often than the program's structure. For this reason, we have divided the programming process into two parts. First, an empty template is created offline as a python script. This template can be seen as a description of an industrial technological process. It contains a set of instructions with defined transitions (i.e., the robot will pick an object and then put it on the table); however, without parameters (i.e., what object will be picked or where it will land). Thus, the template can be created once and later be adapted to conform to different products by setting new instruction parameters.

Supported instructions

The system currently supports the following parametric instructions: *pick from polygon* (to pick up an object from a table), *pick from feeder* (to pick up parts from a

[2] https://github.com/davetcoleman/moveit_simple_grasps

gravity feeder), *place to pose* (to place a previously picked-up object on a selected place on the table) and *apply glue* (simulated gluing). Each of these instructions has specific parameters to be set by the user.

The object type must be set for all of these instructions. For the *pick from polygon* and *apply glue*, a polygon defining the area of interest on the table has to be set so that the user can limit objects of the given type affected by the instructions.

For the *pick from feeder* instruction, a pre-picking pose (see Fig. 5.9c), used for object detection, has to be set using the robot's arm. While executing this instruction, the robot moves to the stored pose, observe the objects with its forearm camera, and picks up the closest object in the direction of the gripper. For *apply glue* instruction, the poses where the glue is supposed to be delivered have to be set using an arbitrary arm of the robot.

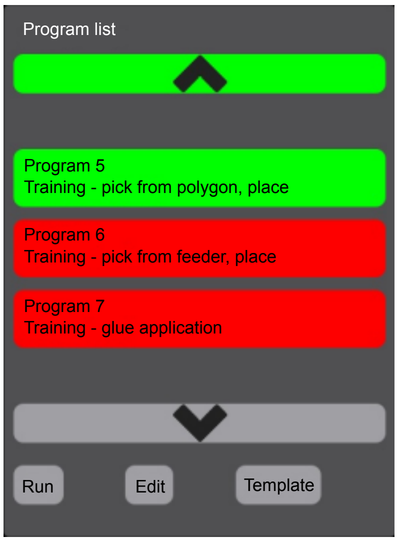
There are also a couple of non-parametric instructions: *get ready*, *wait for user*, and *wait until user finishes*. The first one moves the robot's arms to their default position. The other instructions allow the synchronization of the system and the user. The *wait for user* instruction will pause the program execution until the user is in front of the table, while *wait until user finishes* will pause the program until the user finishes the current interaction with the objects on the table. In our experiments, the behavior of these two instructions was simulated and controlled by the WoZ approach.

User interaction

The interaction between the user and the system is currently achieved using three modalities:

- GUI projected on the touch-enabled surface (which serves as an input for the system and feedback for the user).
- Kinesthetic teaching (input to the system only).
- Sound (feedback for the user only).

The GUI is composed of various widgets. The list of programs (see Fig. 5.8a) shows all the programs stored in the system. It is displayed when the system is in standby mode, i.e., in the case that no program is running or being edited. The color of each entry suggests whether the program has set all the parameters (green) or some of them are not set (red). Only the green programs could be executed. Any program can be duplicated as a new program, with no parameters set or edited, so the user may set or adjust its parameters. During the program editing, the user can see a list of blocks of the selected program and edit a selected block or get

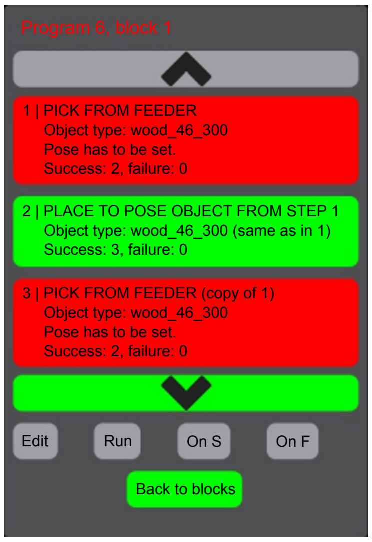

Program list

Program 5
Training - pick from polygon, place

Program 6
Training - pick from feeder, place

Program 7
Training - glue application

(a) List of programs. Green ones are ready to run, red ones need to set parameters.

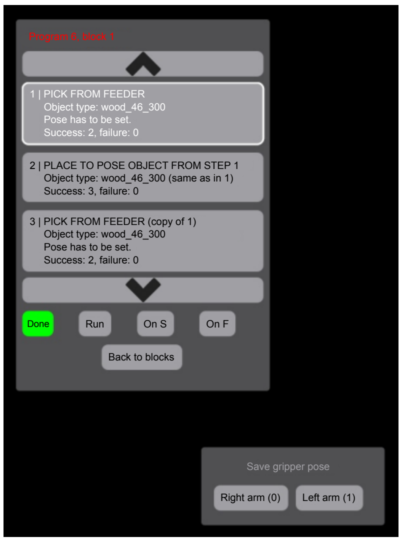

Program 6, block 1

1 | PICK FROM FEEDER
Object type: wood_46_300
Pose has to be set.
Success: 2, failure: 0

2 | PLACE TO POSE OBJECT FROM STEP 1
Object type: wood_46_300 (same as in 1)
Success: 3, failure: 0

3 | PICK FROM FEEDER (copy of 1)
Object type: wood_46_300
Pose has to be set.
Success: 2, failure: 0

(b) List of instructions. Green ones are ready to run, red ones need to set parameters.

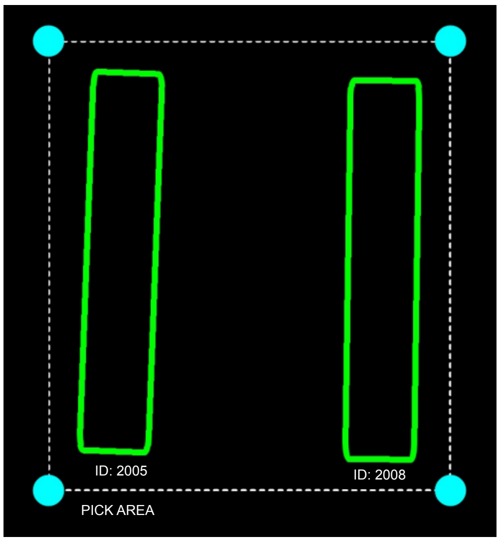

Program 6, block 1

1 | PICK FROM FEEDER
Object type: wood_46_300
Pose has to be set.
Success: 2, failure: 0

2 | PLACE TO POSE OBJECT FROM STEP 1
Object type: wood_46_300 (same as in 1)
Success: 3, failure: 0

3 | PICK FROM FEEDER (copy of 1)
Object type: wood_46_300
Pose has to be set.
Success: 2, failure: 0

(c) A small dialog shows if the robot is able to detect an object in the feeder and allows the user to save the arm pose.



(d) Polygon defining the area on the table from which the objects will be picked up. The green outlines correspond to detected objects.

Figure 5.8: Examples of different widgets from a proof of concept system.

back to the list of programs. The program blocks follow the same color-coding as the programs in the previous widget.

When editing a program block, the list of instructions is shown (see Fig. 5.8b). The selected instruction is always in the middle (except for the first and the last), so the user can see its context (i.e., previous and next instruction). Like the program list, each instruction has a red or green background, indicating whether it has all the parameters set. Once all the parameters have been set, the selected instruction can be executed. Moreover, a gray instruction background suggests non-parametric instruction. There are also buttons to navigate through the program or to select an instruction following either the successful or failed execution of the current instruction. The instruction detail shows:

- the type of the instruction (e.g., *pick from feeder*).
- the parameters (e.g., object type).
- transitions for success and failure (i.e., the ID of the next instruction for success and failure results).

When a program is being executed, the list of instructions differs slightly. All the instructions are grayed out and are not interactive, and the buttons for pausing and stopping the program are displayed instead of the navigation buttons.

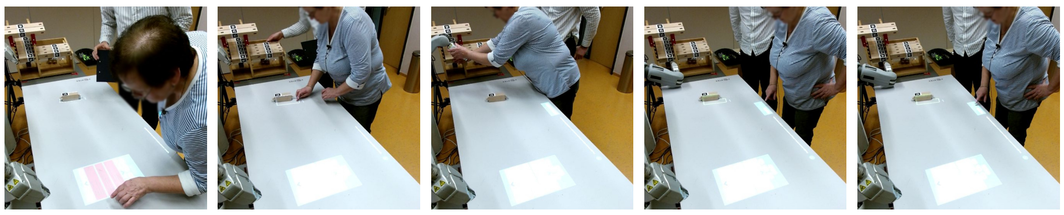
The user is notified about the system's state, the errors, and the currently available actions using a notification bar next to the table's front edge.

The user needs to know the system's state, so for every detected object, an outline and ID are displayed (see Fig. 5.8d). The type of the object is displayed upon clicking on the outline. More information is shown to set the parameters, such as a polygon defining the area on the table or the object's outline showing the position for object placement. The same is also shown during the program execution, so the user knows in advance what object the robot will work with.

Various dialogs exist, which allow the user to specify additional information. For instance, while programming a *pick from feeder* instruction, the user has to specify a pre-pose for object detection by manipulating the robot's arm and then confirming the position using a dialog. The pose is saved after pressing a button corresponding to the arm used (see Fig. 5.8c). The whole procedure is shown in Fig. 5.9 (a-e).

Known limitations

The primary input modality – touch foil – is prone to false readings when metal objects are placed on it, which makes it unsuitable for specific industrial settings. In the future, it might be replaced with or complemented by a vision-based approach



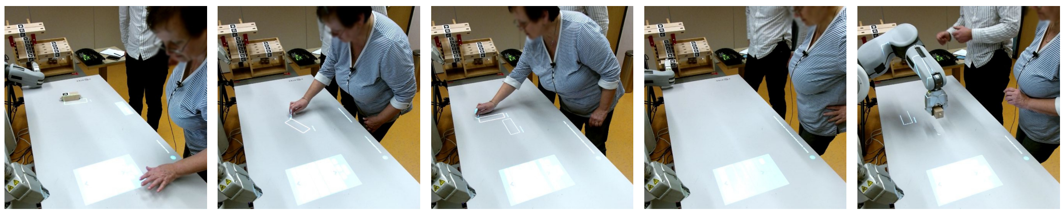
(a) User selects instruction to be set from list (pick).

(b) Object type is set by touching its outline.

(c) Robot arm is used to teach detection position.

(d) Dialog shows if robot is able to detect object in feeder.

(e) User saves position (confirmation sound is played).



(f) User selects follow-up instruction (place).

(g) User adjusts place pose by dragging it on the table.

(h) Another pose, first one also shown for convenience.

(i) User tests *pick from feeder* instruction.

(j) Test of *place to pose* instruction.

Figure 5.9: An example of Human-Robot Interaction (HRI) during the experiment. In this case, the user sets parameters for two *pick from feeder* instructions (one shown) and consequent *place to pose* instructions (both shown). Then, instructions are tested. Two input modalities are used: touch table and robot arm.

(e.g., one from [154]). 3D interaction is currently limited to the kinesthetic teaching of positions, with no means for their later visualization.

5.4 EVALUATION

In order to evaluate the proposed approach and to discover the main usability issues of the early prototype, user experience testing was carried out. Before the experiment, a pilot experiment with three subjects (faculty staff) took place, which helped us verify the prototype's functionality and create the final experimental design.

As metrics, we choose a combination of qualitative and quantitative data. Self-reported data were obtained using a questionnaire consisting of the System Usability Scale (SUS) [26], NASA Task Load Index (NASA-TLX) [54] in its raw form (simplified, with a scale in the range [1..7]) and a custom questionnaire focusing on the specifics of the system. We recorded the task completion times and the number of moderator interventions as quantitative data.

Experiment protocol

The experiment protocol consisted of four phases in total. None of the phases of the experiment was time-limited. There was one moderator in the same room as the participant for the whole length of the participants' track and one operator in a separate room in charge of system monitoring, data recording, and WoZ (used solely to simulate user activity recognition).

1) **Introduction**: At the beginning of the experiment, the participants signed an informed consent form. By the sign, they agreed that the experiment could be recorded for evaluation of the experiment, and the anonymized image data could be used in the research paper or supporting materials. They were told a story about a fictional SME producing wooden furniture: *"The company cannot afford a dedicated robot programmer, so it bought a collaborative robot programmable by any ordinary skilled worker. The robot will assist in preparing the parts for the workers who do the assembly."*

The participants were given information about safety, the workspace parts (interactive table, robot, feeders with furniture parts), and basic interface usage.

2) **Training**: The training phase consisted of three simple programs demonstrating the supported instructions. No specific product was assembled in this phase. The participant set the parameters of each program first, and then the program was executed. During the execution, errors (e.g., a missing object) were intentionally invoked to gain familiarity with the error resolution dialog. In this phase, the moderator proactively helped the participants to complete the tasks and answered all the questions. A short practice of the think-aloud protocol was carried out. Afterward, the participants were told to set the parameters of those three programs independently while thinking aloud.

3) **Main task**: The assembly process of a target product (a small stool) was explained, and the participants assembled it manually. Next, the moderator explained the structure of the corresponding program and the expected workflow. After answering the questions, the participants started working. The moderator was ready next to the participant to answer additional questions but did not proactively help the participant. Once the program was finished, it was started and the participant collaborated with the robot on the task of producing a stool. They produced two stools, after which the moderator told the participants that there was a demand to adapt a product - to produce a higher stool variant. The participant had to stop the running program and adapt its instructions. After the program parameters were adapted, they produced one more stool to verify the adapted program.

part.	gender	age	education	experience with robots	attitude towards new technology
A	F	57	vocational (technical)	none	skeptical
B	M	46	secondary (technical)	seen robot at least once	neutral
C	F	27	secondary (economics)	none	neutral
D	M	33	secondary (technical)	seen robot at least once	early adopter
E	M	24	secondary (technical)	works on workplace with robots but not next to them	neutral
F	M	34	undergraduate (technical)	none	skeptical

Table 5.1: Demographic data of the participants.

4) Feedback: After finishing the tasks, an open discussion took place. The participants were asked for their impressions or additional questions. Lastly, they filled in the questionnaire.

Stool assembly

The intended workflow of the main task is that the user does the assembly while the robot prepares the parts needed in the next steps "on background". The program is divided into three blocks (see Fig. 5.10). Blocks 1 and 2 have the same structure and serve to prepare the parts for the sides of the stool (two legs, two connecting parts, application of glue). The purpose of two blocks is that the user might set parts within one block to be supplied from, e.g., the left feeder and the other block from the right feeder. Block 3 serves to prepare the connecting parts for the final assembly of the sides of the stool.

Participants

In cooperation with an industrial partner (ABB Brno), we have selected six regular shop-floor workers of various ages, genders, and technical backgrounds (out of 27 volunteers) to participate in our study. These participants are labeled as Participants A, B, C, D, E, and F. Five work in quality control; one (E) works as a mechanic. The participants' demographic data are presented in Table 5.1.

5.5 RESULTS

The section provides the results of the experiment.

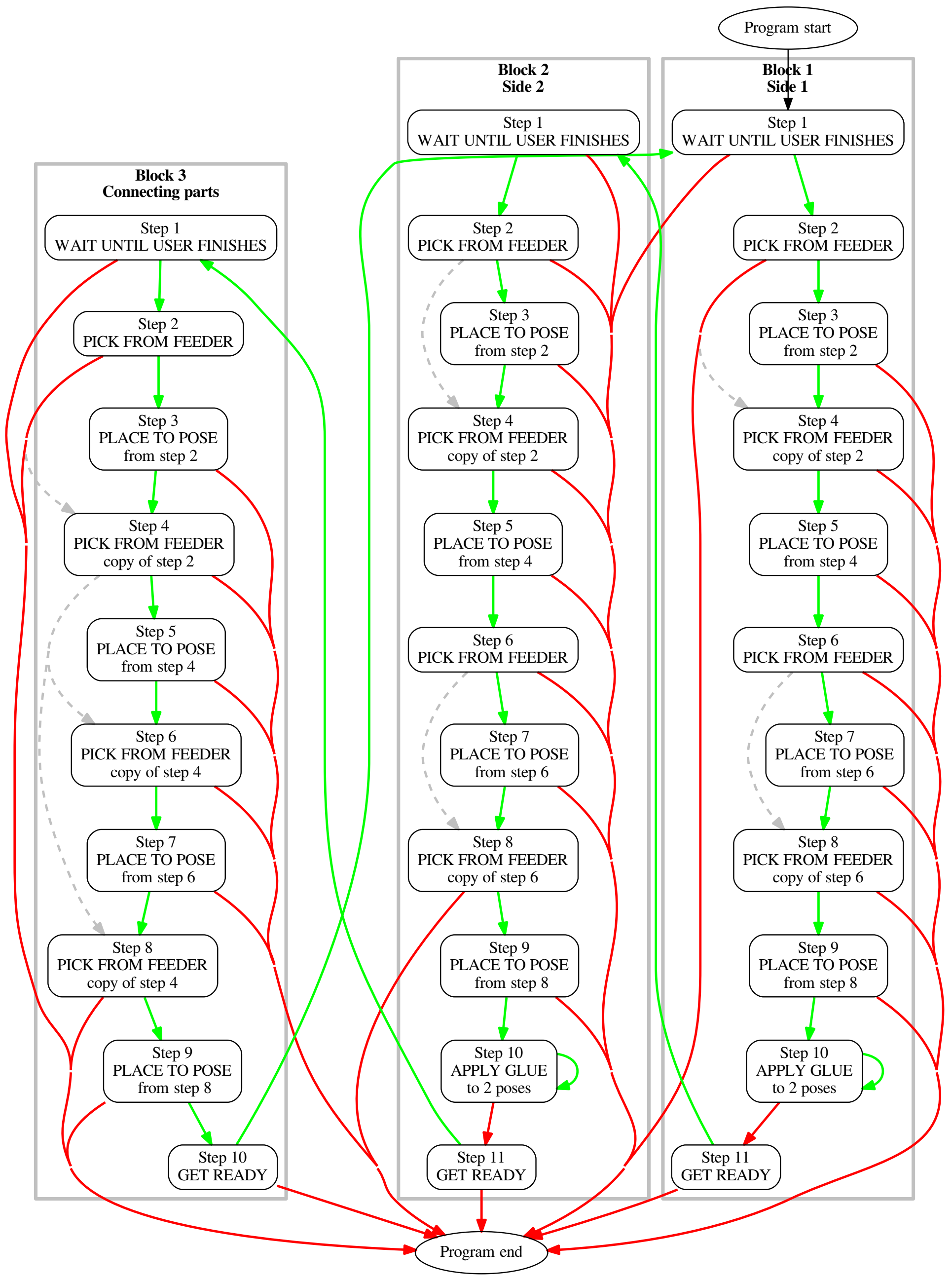


Figure 5.10: Stool production program. The green edges represent *on_success* transition, while the red ones represent *on_failure*. The grey edges show dependencies. In the case of *apply glue*, there is a loop. The robot applies glue to one object in a specified area. If an object is found, the program flow continues to the *on_success* instruction - it tries to apply glue to another object. If there is no object with no glue applied, the flow continues to *on_failure* (next instruction).

Qualitative and quantitative data

Table 5.2 shows the results per participant. The mean time to complete the main task was 2711 s (SD 620 s) with 11.7 (SD 6.7) moderator interventions. The main task consisted of setting the following instructions: 5x *pick from feeder* (2 parameters), 12x *place to pose* (1 parameter), 2x *apply glue* (4 parameters), resulting in settings of 30 parameters in total. The mean time for program adaptation task was 1053 s (SD 215 s). It consisted of setting: 2x *pick from feeder*, 2x *apply glue*, and optionally, adjustment of place poses (based on previously set poses), resulting in at least 12 parameters in total. These times include the delays caused by system errors (unreliable object detection and unstable manipulation pipeline). The mean SUS rating was 75.8 (SD 8.9), while for comparison, the system from [62] scored 66.75 (SD 16.95). The mean NASA-TLX was 33.3 (SD 8.8).

From the custom questions (see Table 5.3), it seems that the participants, in general, liked interacting with the system and felt safe; however, they were confused from time to time. However, during the experiment, in most cases, it was enough to tell them to check the notification area, and they could continue afterward.

Programming

Observation of the users revealed that the current visualization of the robot program is probably insufficient. It often took considerable time to realize what instruction was currently being programmed, especially for the case of repeating sequences of program items (e.g., *pick from feeder*, *place to pose*, *pick from feeder*, *place to pose*).

Not entirely consistent terminology (e.g., program instruction was sometimes referred to as an item and sometimes as a step) may have contributed to this. Probably because of the similar appearance, for some participants, it was difficult at the beginning to distinguish between a program block and a program instruction.

Probably the most common issue during programming was the participant forgetting to press the *Edit* button to switch from the view-only mode to the parameter settings mode for the selected instruction. The participants often tried to adjust, for example, place pose and were confused as to why it was impossible. Also, it was often unclear that it was only possible to execute individual instructions and not the whole block. Initially, two participants thought that the instructions (displayed in the program visualization) were for them, so they should perform, e.g. *pick from feeder*. One participant asked if there were also assembly instructions for the workers.

There have been cases where the user accidentally changed the selected object type. Although this was covered during training, some participants thought that the object type was selected when they put an object of that type on the table. Although the objects of a selected type were highlighted differently (with a green outline), most of the participants only guessed what type was selected or checked it in the program visualization where the information was in textual form.

Individual instructions

1) Pick from feeder: Participants were often confused, as it was required to select the object type on the table and then to use a robot arm to set the pose enabling the detection of parts in the feeder. We noticed several cases the participant tried to select an object by knocking on it (instead of clicking on its outline), both the object on the table and in the feeder. The participants commonly skipped the object selection, grabbed the robot arm, and tried to set the pose, even above the object on the table, even though they were learning picking from a gravitational feeder. After pressing *Edit*, dialog buttons for saving the arm pose (grayed-out at the time) were sometimes used "to select arm" before any other interaction. Most users took a new part from the feeder and put it on the table when they needed to select the object type even though there were already objects of that type that could have been used for this purpose. When adapting the program, it happened twice that the participant accidentally set the position for the other feeder (e.g., the instruction initially used the left feeder, and they switched to the right one). This would mean that the robot would not be able later to place the object, as the following place pose (on the opposite side of the table) would be out of its reach.

2) Place to pose: Common sources of problems were unreachable place poses, or places too close to each other, preventing the robot from placing parts successfully. The only possibility was to find out by trial and error. For all the participants, it was difficult initially to handle separated translation (by dragging) and rotation (using a pivot point). Some intuitively attempted to use multi-touch gestures (not supported by the interface so far), including one participant who does not own any touch devices. Although the initial position of the placing pose was in the middle of the table, some participants had trouble finding it, especially if there were many objects around. Some tried to drag the outline of a detected object or placed an object into the outline of the place pose. Visualization of the place poses from other instructions (differentiated by a dotted line and a corresponding instruction number) were confused a few times with the current place pose, and the users tried to move them.

Measure	**A**	**B**	**C**	**D**	**E**	**F**
System Usability Scale	87.5	67.5	77.5	75.0	85.0	62.5
Simplified TLX	25.0	33.3	30.6	22.2	41.7	47.2
time to set program (s)	3849	3025	2618	2217	2661	1897
interventions	21	7	20	12	6	4
time to adapt program (s)	1088	1447	1118	958	738	968
interventions	11	4	12	2	2	2

Table 5.2: Qualitative measures, task completion times (stool program) and number of moderator interventions (including answering questions).

For successful collaboration with the robot, it was necessary to organize the workspace so that the robot could prepare the parts for the next steps while the user did the assembly. Only Participant B explicitly thought about the organization of the workspace. The others had minor problems with it or required help. Participant C placed the parts in a very chaotic way. During the training, participants were explicitly told that they might move widgets (e.g., program visualization) across the table; however, most of them did not use it and rather adjusted the place poses so that they did not collide with the widget.

3) Glue application: The most common issues were object type selection (attempts to select using the robot's arm) and difficulties with the number of actually stored poses (shown textually). The fact that it is necessary to store required poses only with regard to one object and that the robot will do it the same way for other objects in a given area was also generally unclear.

Program execution

During the program execution, errors occurred relatively often, especially when the robot tried to place an object; erroneous detection prevented it from doing so. In the event of an error, a dialog appeared, and the sound was played. Most issues were solved by pressing the *Try again* button. The participants were explicitly told to pay attention to errors. Some participants reacted immediately, others after some time, and one seemed to ignore the errors and had to be told to solve them. Once in a while, it was necessary to warn a participant that he or she was blocking the robot by occupying part of the table where the robot was meant to place parts.

Statement	A	B	C	D	E	F
Collaboration was effective.	5	4	5	5	4	4
I felt safe.	4	5	5	5	5	5
Robot motions were uncomfortable.	2	1	1	1	1	1
It was easy to see what the robot was about to do.	4	5	5	4	4	2
The robot hindered me at work.	1	2	1	1	1	1
I watched every movement of the robot.	3	1	2	3	4	2
Learning the robot using its arm was intuitive.	4	4	5	5	5	4
Learning the robot using the interactive table was intuitive.	4	4	5	5	5	3
Interactive table shows all necessary information.	5	2	5	5	5	4
Sometimes I did not know what to do.	5	5	4	2	4	4

Table 5.3: Custom questionnaire, 1 - totally disagree, 5 - totally agree

General findings

To our surprise, no participant complained about imperfections of the projection (shadows, inaccurate registration), low text readability, and interface response times. Each participant had an issue at least once with a non-touchable margin of the interactive table, which was not indicated by the projected interface. There were also issues with pressing the buttons twice, where users tried, for example, to select an instruction that was immediately unselected. Although the inactive buttons were grayed out, most users tried to press them anyway when they thought they should work.

There was considerable visual clutter with many objects on the table or during the stool assembly. Interestingly, no one mentioned it. Difficulties with moving interface elements (e.g., place pose) across longer distances were observed, especially if many objects were on the table. Again, no one complained or asked if there was an alternative method to dragging.

There were sounds (confirmation, warning, error) as a complementary modality. Only Participant B explicitly appreciated it. Regarding safety, only Participant A once noted that a particular movement was probably not safe. No one used the emergency stop button.

5.6 CONCLUSIONS

In this research, we targeted problems of the existing solutions in the interaction between human workers and industrial collaborative robots, particularly in the context of programming robots in SMEs. The proposed and evaluated interaction system attempts to reduce the mental demands and attention switching by centering all interaction elements in the shared workspace. It is achieved by the interactive SAR (combination of projection and a touch-enabled table) and kinesthetic teaching, in accordance to the O1.

Non-expert users program a robot on a high level of abstraction and work within the task context, free of any additional external devices and with immediate visual feedback. As required by the O3, the user interface supports the programmer with the information of instruction's spatial parameters and provides them with a simple method of setting these parameters.

To fulfill the O4, we have conducted the user experience, which proved the potential of our concept when all six regular shop-floor workers were able to program the robot to prepare parts for a stool assembly, collaborate with the robot, and adapt the program for an alternative product within a reasonable time.

We found no fundamental issues during the experiment forcing us to reconsider the approach. However, the task state awareness, in particular, must be improved and support the workspace layout. The participants rated the system positively despite several minor usability issues and system errors caused by its experimental nature.

6 END-USER ROBOT PROGRAMMING CASE STUDY: AUGMENTED REALITY VS. TEACH PENDANT

A significant problem emerges with novel and unusual interfaces and interaction methods: how to compare them with existing and well-known methods or interfaces. While comparing partial interfaces' features might be easy and intuitive, comparing two complex and highly different systems is challenging in terms of experiment design and evaluation of the results. Experiments with novel interfaces could provide good insight into whether the interface is usable by measuring subjective data. A fair comparison with the existing method is crucial for measuring improvements in, e.g., efficiency, to prove that the new method offers added value over the existing one and could be successfully deployed in real-world industrial settings.

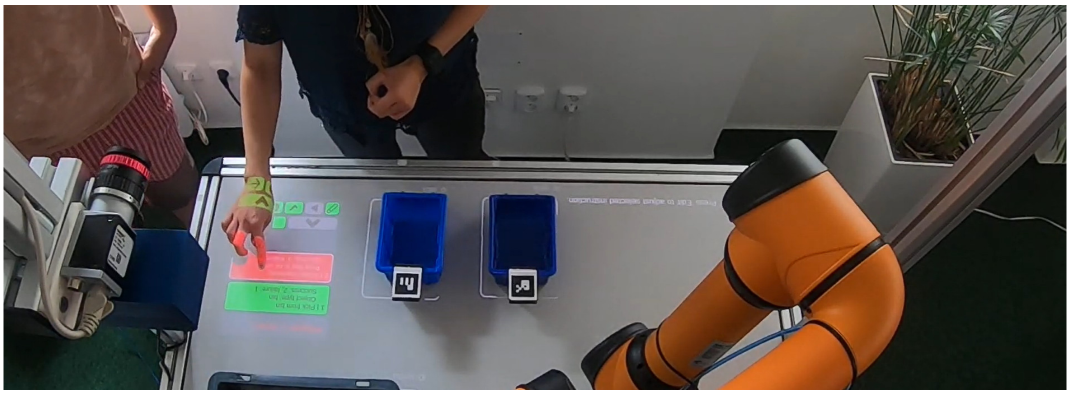


Figure 6.1: Participant programs a visual inspection task using the ARCOR SAR interface.

Several experiments were conducted to evaluate usability of our AR interface ARCOR (see Fig. 6.1) for end-user robot programming [13, 91, 92]. This interface allows the user to program the robot using highly abstracted instructions such as *PickFromTable* or *DrillHole*, using a user-friendly graphical interface projected on a touch-enabled table. Although we evaluated the interface several times, no comparison with any existing method has taken place, as our system did not support any standard industrial robotic arm. Recently, we added support for the Aubo i5 robotic arm. This chapter presents a preliminary experiment designed as a case study to get insight into comparing such different interfaces.

This chapter is an extended version of my published paper [68].

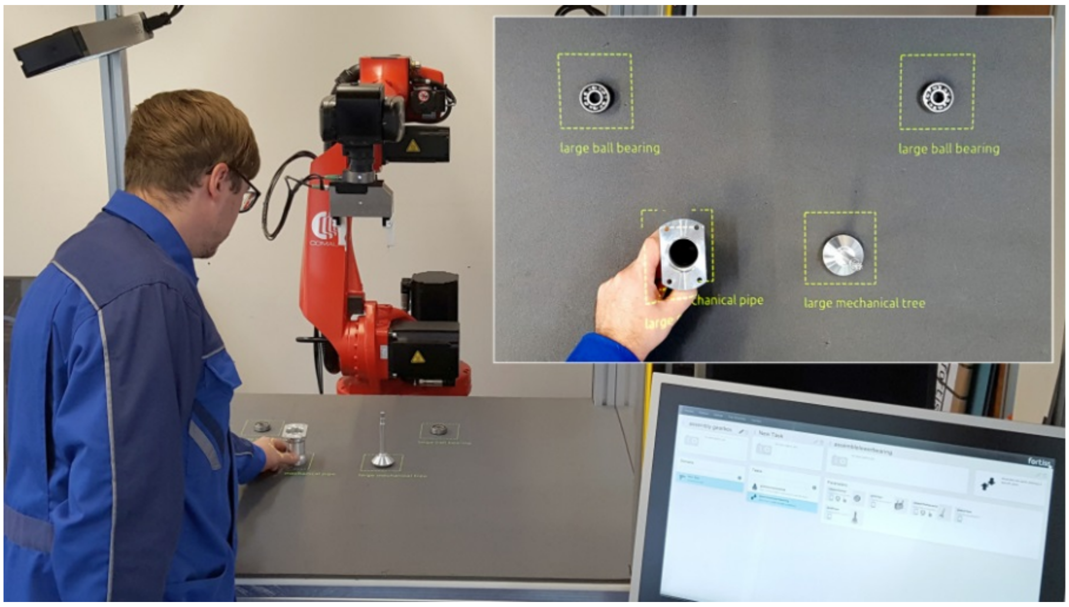


Figure 6.2: Cognitive robotic work cell with features such as object recognition, projector-based highlighting, hand and body gesture recognition, and touch-based graphical interface for the task-level instruction of the robot. Reprint from [106].

6.1 RELATED WORK

One of the traditional methods of programming industrial robots is through the teach pendant. There exist various pendant interfaces. Some of them, such as ABB FlexPendant with its text-based programming, are targeted to expert users. In contrast, others are more suited for less skilled users, such as Universal Robots Polyscope's tree-based program visualization and wizards. Emerging alternative methods aimed at simplifying the robot programming for non-experts were often not evaluated with a (user-friendly) pendant as a baseline method. Only a few examples of evaluations exist where such comparisons have been carried out. However, the published experiments have various limitations.

For instance, Perzylo et al. [106] proposed a projected SAR interface using a cognitive robotic work cell (see Fig. 6.2), enabling object-centric robot programming for non-expert users. The work cell was capable of object detection and classification, object highlighting using projection, recognition of hands and body gestures, and touch-based user interface. They evaluated the initial concept with one expert user. Stenmark et al. [136] proposed another end-user robot programming tool, which was evaluated with 21 non-experts; however, the comparison with the traditional tool was only evaluated with one expert.

The CoBlox, proposed by Weintrop et al. [149], using a Blockly-based programming environment, was evaluated with 67 non-programmers but only in simulation. The experiment with a PATI interface carried out by Gao et al. [46] seems well designed, with a sufficient number of participants, however only with a simple pick and place task. Existing experiments are usually designed ad hoc, as there is a lack of proven methodology. For instance, a method to compare HRI approaches is proposed by Rodamilans et al. [115]; however, extension beyond the trajectory teaching task would be needed.

6.2 EXPERIMENT DESIGN

A preliminary 2-condition within-groups case study was conducted. The main goal of the presented case study was to verify that the proposed method of simplified robot programming is suitable for a visual inspection task and performs better than the teach pendant (whose interface is similar to UR Polyscope). The robot is instructed to pick the bottle opener from the table, put it in front of the camera, trigger the inspection method, and, based on the inspection result, put the bottle opener on one of the boxes on the table. A few high-level functions such as pick, place, or suction (on/off) were prepared in advance in the pendant to make the comparison fairer. The experiment was conducted with three participants (two males and one female) in a lab-like environment. All participants had little, or no prior experience with AR, and participants A and B had little or no experience with the teach pendant. In contrast, participant C had moderate prior experience with pendants.

The experiment involved two sessions (first with pendant, second with ARCOR) of training and programming the actual task. Each of the sessions was followed by filling in the standard questionnaires, namely SUS [26], NASA-TLX [54] and User Experience Questionnaire (UEQ) [129], and discussion. We recorded the participants using a standard camera for future analysis. Moreover, several physiological data were recorded using the Empatica E4 wristband.

6.3 RESULTS

All participants completed the task using both methods (teach pendant, ARCOR). For each participant, the time needed for both introduction and programming was lower for the ARCOR interface (see Table 6.1). The ARCOR also performed better in usability, UX, and task load metrics. Detailed cases for each participant follow.

Participant A

The first participant was a 25 years old male who works as a programmer in a software developing company. While using the pendant, the moderator had to intervene approximately eight times to help the participant overcome the issues

Participant	A_p	A_a	B_p	B_a	C_p	C_a
Introduction [s]	359	179	449	311	785	174
Task [s]	562	189	749	309	510	146
TLX [0, 100]	72.22	36.11	44.44	27.78	33.33	19.44
SUS [0, 100]	52.50	82.50	42.50	80.00	70.00	90.00
UEQ_{ATT} [-3,3]	-1.17	2.00	-0.17	1.83	1.83	2.50
UEQ_{PRA} [-3,3]	0.25	2.08	-0.50	1.83	1.58	2.25
UEQ_{HED} [-3,3]	-0.25	2.12	-1.25	1.62	0.25	2.00

Table 6.1: Durations of introduction and programming for both (p)endant and (a)rcor modality. Subjective metrics for each participant and both modalities. Higher means better for all subjective metrics except NASA-TLX.

with the pendant interface, mainly finding the right buttons for the desired task. The participant was a bit frustrated when he wanted to copy a block of instructions, which was impossible.

With ARCOR, only one moderator intervention was necessary when the participant overlooked the dialog for saving the robot position. Sometimes, the participant was unsure what is the next required step, but he was always able to resolve this uncertainty using the notification area of the interface. The participant complained about the positioning of some GUI elements, which were sometimes hidden by real objects.

The participant considers the teach pendant too complicated, slow, and cumbersome. He prefers the ARCOR interface because many things are already prepared in advance, allowing him to focus on the programming itself.

Participant B

The second participant, the 41 years old male working as an application tester, struggled with the complex GUI of the pendant: there were difficulties in finding buttons, instructions, and instruction lists. This was the leading cause of frequent moderator interventions. Moreover, the participant asked the moderator several times whether he was proceeding correctly in setting individual instructions and waypoints.

When the participant was using the ARCOR interface, there were significantly fewer moderator interventions related only to the touch surface problems (e.g., non-registered touches). The participant successfully used the notification area of the interface when he felt lost or did not know how to proceed further.

Although the participant preferred, based on the results, the ARCOR interface better, there were some complaints about setting the box location area, where the interface could be more automated and, for example, not allowing the user to move the UI elements outside of the touch-enabled surface. The participant considered the pendant approach difficult but admitted that it could be learned if there was no other option.

Participant C

The last participant was a 23 years old female who works as a programmer. This participant's prior experience with the pendant is reflected by the lowest time needed for an introduction to this modality. It could explain better scores in all measured metrics compared to other participants. However, she still ranked the ARCOR modality better in all metrics. Despite the prior experience, the participant was insecure at the beginning and was using quite a large amount of help from the moderator. After a few minutes, however, she became more certain about various interface elements.

For this participant, setting the robot's position was physically challenging, which could be one of the reasons why the ARCOR interface was ranked better, as it requires less direct manipulation with the robot. The participant had no fundamental problem with the ARCOR interface; she only suffered from some design issues like ambiguous buttons, visualization of inactive buttons, or slow response from the system, where she was uncertain whether she pressed some button successfully, for instance.

She felt good using both interfaces, but she considered the ARCOR interface simpler and faster.

6.4 conclusions

The preliminary experiment was focused on comparing two highly different methods of robot programming: the first one utilizes the SAR for visualization and programming, and the other uses the user-friendly teach pendant. It was necessary to deal with different complexity, levels of abstraction (high for SAR, low for pendant), and specifics of each method. The results indicate the potential of the ARCOR system, which the participants preferred over the pendant and required less time to train and program the visual inspection task.

Part III

MOBILE AUGMENTED REALITY AND SPATIAL ROBOT PROGRAMMING

The spread of touch-enabled mobile devices with a large screen and users' familiarity with them persuade me to explore the possibilities of mobile AR in end-user robot programming. We have found that despite the need to hold the device during the programming phase, the high usability and rich possibilities of visualization of 3D spatial information in task space make it a strong candidate for an ideal end-user programming device. The following chapters present a novel method called Spatially Anchored Actions and its implementation into a fully functional end-user robot programming tool on a mobile device. The method was evaluated with more than 20 participants. It was shown to significantly improve the program comprehension for the users over a standard end-user robot programming method while maintaining similar usability and temporal demands.

7 SPATIALLY SITUATED END-USER ROBOT PROGRAMMING IN AUGMENTED REALITY

The previously presented concept of robot programming using the projected SAR suffers several drawbacks. Mainly, the projection itself is only two-dimensional (assuming the projection on the table); therefore, it is difficult or impossible to visualize any 3D spatial information in the robot's task space. Moreover, using a single projector causes the objects on the table to drop shadows, thus creating holes in the projected image. It could be partially solved using multiple projectors, which implies new problems in terms of spatial demands, higher cost of the solution, or mutual calibration of the projectors. Moreover, the second scenario, proposed in Section 3.2 involves several scene objects, such as a testing device or various boxes for storing the PCBs, which occupy the task space and interfere with the projection and touch surface integration.

The only way to set precise 3D spatial information using the previously proposed SAR interface was to manipulate the robot's arm manually. In seeking a solution, able to visualize 3D spatial interaction and interact with 3D virtual objects and widgets in AR, two approaches emerged. The first is the HMD-based AR and the other is the mobile AR. As was described in Section 2.3, the HMD's advantage is the ability to visualize spatial information in a person's field of view without the need to hold any device in hand and to provide an interaction with the virtual content using natural ways, although possibly cumbersome in industrial settings, such as hand gestures or voice commands. The advantage of the mobile AR is the relatively cheap customer-grade device (smartphones and tablets), which is well-known to the users. On the other hand, it forces the user to hold the device in their hands, which restricts the usage of this method in some applications.

I proposed an experimental system to overcome the abovementioned problems according to the stated objectives. The requirements for a new type of user interface are defined: integrate the programming tool into a real 3D environment, use scene and object knowledge to reduce user mental load, visualize a program stage, and robot's knowledge of the environment to improve the user's feedback. I proposed a new concept of robot programming in a three-dimensional environment, shared by the programmer and the robot, using AR on a mobile device (O_1). The program's flow is defined as a sequence of individual actions, which forms a directed acyclic graph (O_2). This representation allows conditional and parallel execution. The actions are visualized in their spatial context, i.e., in the place where they are

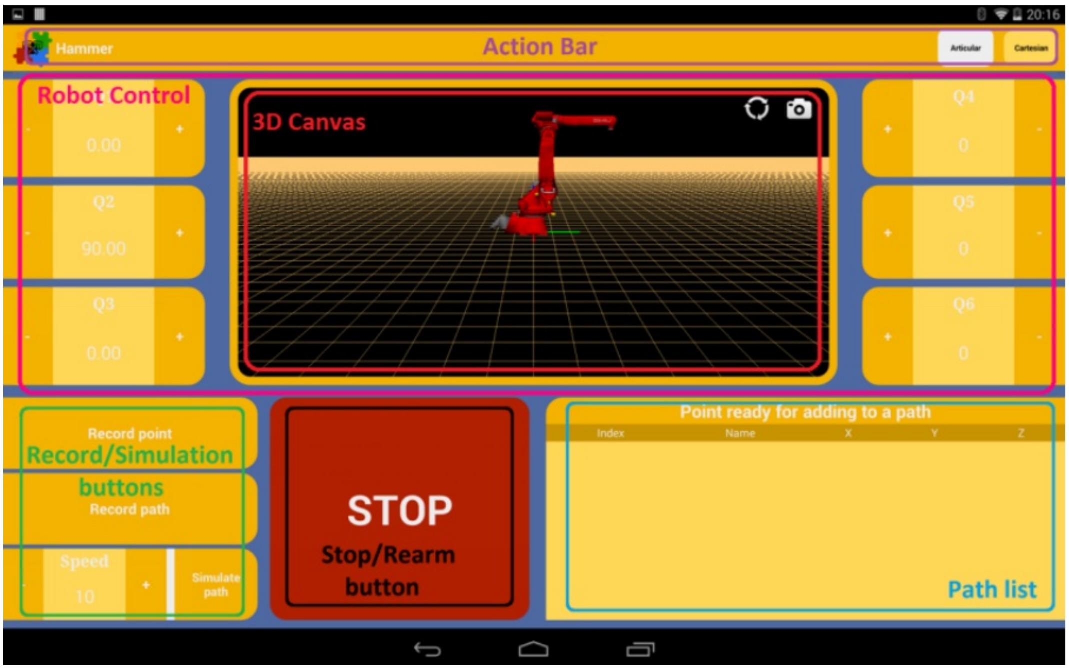


Figure 7.1: Cognitive robotic work cell with features such as object recognition, projector-based highlighting, hand and body gesture recognition, and touch-based graphical interface for the task-level instruction of the robot. Reprint from [90].

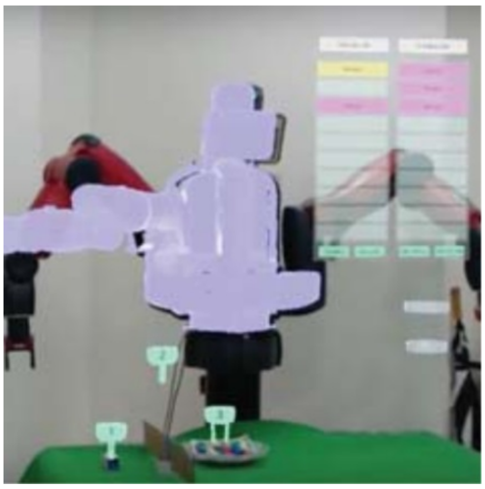
executed. The method combines the semantical knowledge of the environment and 2D and 3D widgets to allow the programmer to define the program's spatial parameters (O3). Last but not least, the proposed method was evaluated with seven non-programmers (O4). The experiment has shown the strong potential of mobile AR usage in end-user cobot programming.

The following chapter is based on research published in paper [67]. The research presents the application of the principle of bringing the human-computer interaction to the task space in the design of a new tool for cobot programming.

7.1 RELATED WORK

An increasing number of collaborative robots in SMEs requires searching for new methods for end-user robot programming. Various techniques incorporating the AR were proposed, mostly based on visual programming [90, 158], PbD [8, 79] or combination of both [62, 91]. These methods may differ in both input and output modalities and utilize the AR for programming and giving visual feedback to the user.

Mateo et al. [90] presents a Hammer, novel tablet-based user interface for industrial robot programming (see Fig. 7.1). They used the Scratch programming language to create an Android application, primarily focused on end-users programming the polishing, milling, or grinding operation. It allows both to program


(a) A screenshot from the MR perspective of a user programming a robot motion. Users can specify green waypoints.

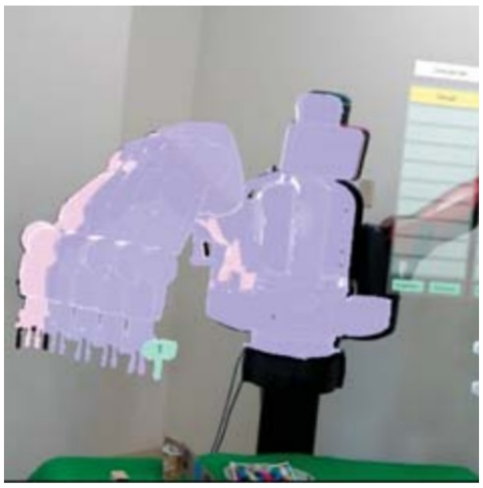

(b) After creating the waypoints, users can visualize the robot arm motion that is planned through the waypoints.

Figure 7.2: Mixed reality interface. Reprint from [158].

the robot using visual programming, allowing operators with limited programming knowledge to create or adapt programs easily. Besides, the application offers to control the robot using a teach-pendant-like interface or monitor the robot using the AR features.

Gadre et al. [158] proposed a Mixed Reality (MR) system for robot programming using HMD (see Fig. 7.2). The system enables the user to define a robot motion using the waypoints. The interface is controlled using a combination of hand gestures and a 3D WIMP[1]-like interface. They compared this system against a 2D keyboard and mouse system for programming pick & place tasks. Gadre et al. [158] found that novice users were significantly faster and better able to successfully program the robot using the MR interface than the 2D keyboard and mouse interface. Moreover, participants reported a lower cognitive workload using the MR interface. They appreciated the possibility of simply moving their head or body to readjust the point of view, compared to a rather unintuitive movement with a virtual camera using the mouse.

Quintero et al. [111] designed an AR system using Microsoft HoloLens HMD capable of 3D robot trajectory specification (see Fig. 7.3), virtual previews of robot motion, visualization of robot parameters and online reprogramming during simulation and execution. As trajectory programming is a core task in industrial robot automation, their system allows two modes of trajectory programming: free space trajectories and surface trajectories. Both methods utilize the user's gaze, hand gestures, and speech recognition to define the trajectory. The former allows the

[1] Windows, Icons, Menus, Pointer

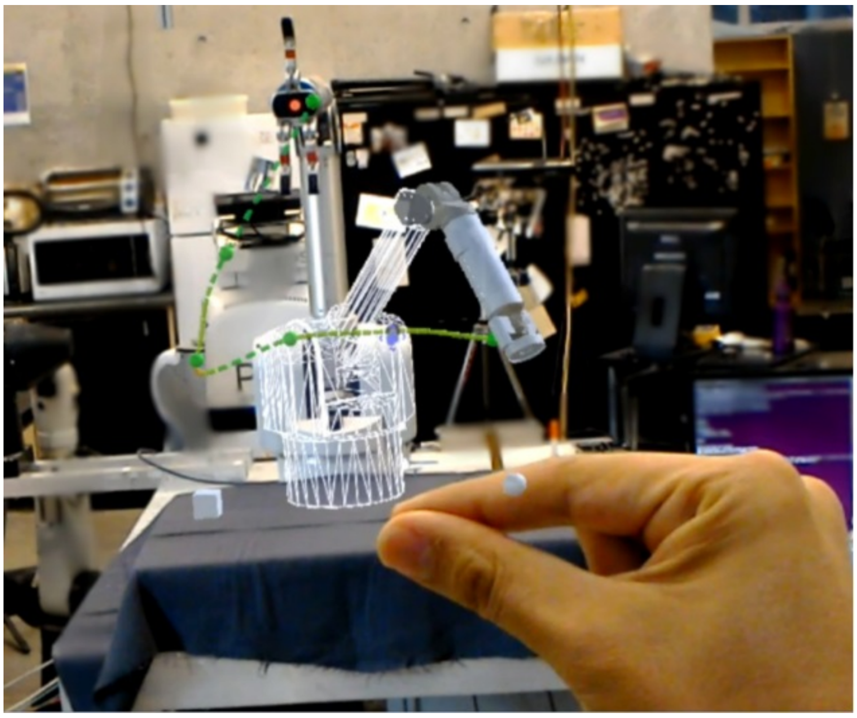


Figure 7.3: A user's point of view from within the Hololens: The user creates, modifies, simulates, and executes a pick-and-place trajectory using our AR robotic system. Notice that the holographic 7-DOF robot arm overlays the real robot and permits the user to simulate motions before execution. All interaction is performed through speech and gestures. Reprint from [111].

operator to create arbitrary trajectories in the free space while the system provides the user with the visualization of the trajectory and simulation of robotics motions. The latter benefits from the surface analysis, either using the 3D reconstruction or CAD model, and allows the user to define the waypoints directly on the surface. Using the surface knowledge, the robot's end-effector is automatically perpendicular to the surface and applies a constant force on the surface. Quintero et al. [111] compared the system with kinesthetic teaching and found out that it takes less teaching time for the users to use the proposed AR interface and, at the same time, shows better performance when specifying a complicated path (e.g., sine curve).

Blankemeyer et al. [21] present another HMD-based system using Microsoft HoloLens for assembly task programming. The system detects all assembly components on the table using the optical markers and provides the user with their virtual counterparts. The user carries out the assembly step virtually by moving the virtual components to the desired position, using standard Microsoft HoloLens interaction elements. The system records the initial and final positions of the parts for each of the assembly steps. The robot is then able to recreate the assembly using the real components.

Stadler et al. [134] explored the effect of showing task-based information using the tablet-based AR on the perceived workload of robot programmers. They

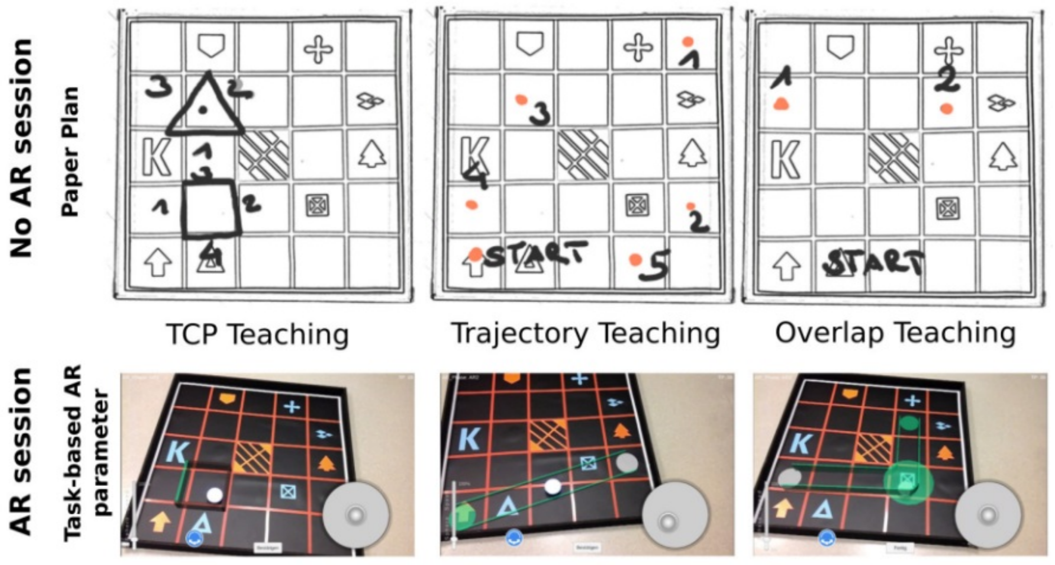


Figure 7.4: Visualization of task-based parameters per task (TCP, trajectory, overlap) in the No AR session (paper plan, top) and AR session (bottom). Reprint from [134].

conducted a study with 19 professional industrial robot programmers, including novices and experts. The participants had to execute three typical robot programming tasks with a Sphero robot (see Fig. 7.4) – tool center point teaching, trajectory teaching, and overlap teaching – using the tablet interface. Once with AR task-based information and once without this supportive information. Using the NASA-TLX questionnaire, they found out that visualizing the task-based parameters using AR decreases the mental workload of the participant, but, at the same time, it increases the task completion time. Moreover, they speculate that the superimposed task-based information relieves the operator's memory, as he or she does not have to remember the teaching points and directions.

Magnenat et al. [85] have shown that the operator's overall performance could be significantly increased by incorporating AR and visual feedback into a tablet-based system for robot programming.

Recently, several solutions based on tabletop projections emerged. We have proposed a SAR system using a table with a touch-enabled surface and projector above the table, projecting both User Interface to program collaborative robots and showing contextual information of objects on the table and the state of the system.

Gao et al. [46] provided another tabletop SSAR solution called PATI (see Fig. 7.5) for industrial end-user robot programming of manipulation tasks. They have proposed an illustration-based language with different visual elements, which could be defined or manipulated by gestural input using common touch screen gestures. The user's gestures are detected and classified using computer vision in combination with the Kinect2 camera. The language provides several selection tools (for object selection or area definition), action tools (actions the robot should perform,

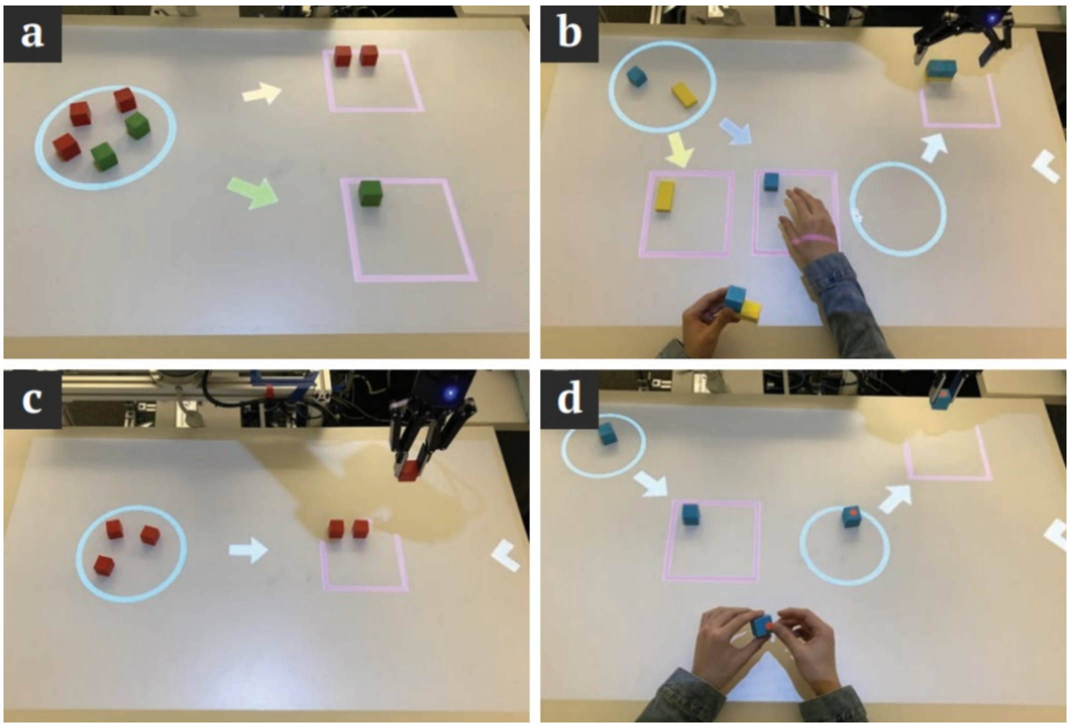
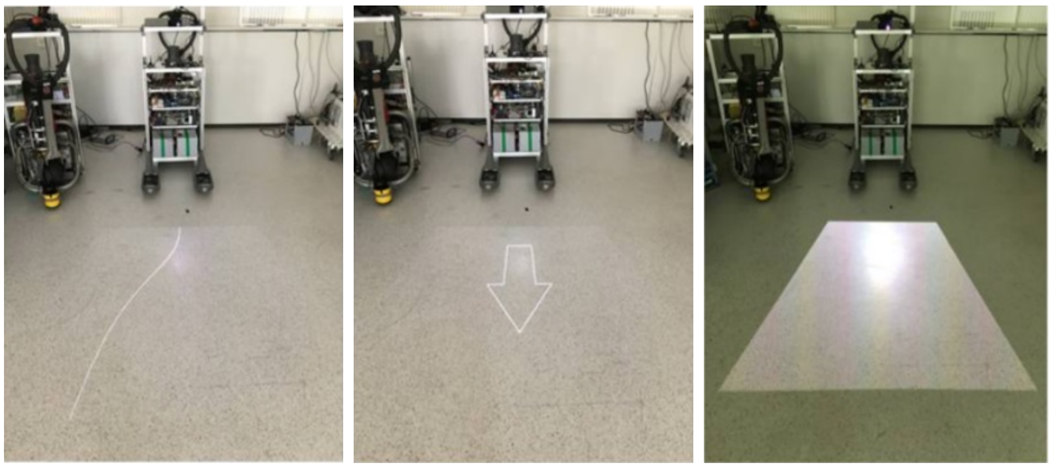


Figure 7.5: Possible applications of PATI include (a) sorting, (b) assembly, (c) alignment, and (d) inspection. The colored arrows in (a) and (b) represent how objects would be sorted based on color attributes. Reprint from [46].

such as moving objects from one area to another), and attributes (properties of selection or action tools). By combining the tools mentioned above, the user can compile various types of programs (see Fig. 7.5).

To investigate the effects of presenting the robot's intentions to the human, Bunz et al. [28] conducted an experiment involving a mobile robot with a projector mounted on top of it, projecting various patterns indicating its intended movement (see Fig. 7.6). The experiment has shown that although the type of projection influenced the participants' attention, it had no effect on their path selection. They speculate that the participants struggled with understanding the precise meaning of the projected patterns and that they should be designed more thoroughly next time. Moreover, the experiment's design significantly constrains the participants' possible movements, which could also influence the results.

Head-up displays and projected user interfaces benefit from freeing the operator's hands, enabling direct manipulation of real objects. On the other hand, contemporary head-up displays such as Microsoft HoloLens and others suffer from a narrow field of view and potential users' discomfort in long-term usage. Moreover, end-user programming systems based on hand-held AR overcome head-up-based systems in terms of speed and user experience [13]. While projected interfaces do not suffer from these issues, they are not currently able to present information in free 3D space and are only suitable for tabletop scenarios [91].



(a) Projection A: trajectory line (b) Projection B: arrow (c) Projection C: white area

Figure 7.6: The different intention communication modes: Four types of intention communication were tested, three of which are depicted above. The fourth mode is without projection. Reprint from [28].

The AR systems often benefit from the knowledge of the environment and therefore offer new possibilities in end-user programming. The spatial situated programming incorporates real objects into the programming process. For instance, Ivy [39] (see Fig. 7.7), an immersive VR application, enables users to link different smart devices, create automated behavior based on readings from smart sensors and visualize data flows between those devices. Ens et al. [39] conducted an explorative study with eight professionals with expertise in IoT[2], Information Visualization, and Computer Graphics to identify limitations and opportunities of the application. The initial study has shown that the user interface is understandable, and the participants had no fundamental troubles with it. However, they have noticed some limitations in selecting objects at further distances. Some participants saw the immense potential in presenting spatial information, such as virtual connections between physical devices, using immersive visualization.

Reality editor [56] is another example of spatially situated programming, enabling programming of behavior and interactions of smart objects, using hand-held AR devices. In the proposed approach, the tablet-based AR is combined with semantic information of the objects on the table to enable regular shop-floor workers to create robotic programs. Contrary to solutions mentioned above [46, 91, 111, 134], my system aims at both defining the flow of the program and setting its parameters. Usage of relatively cheap mobile devices can significantly lower the cost of the solution compared to approaches using high-end HMD devices [111, 158] while remaining more flexible than projection-based solutions [46, 91].

[2] Internet of Things

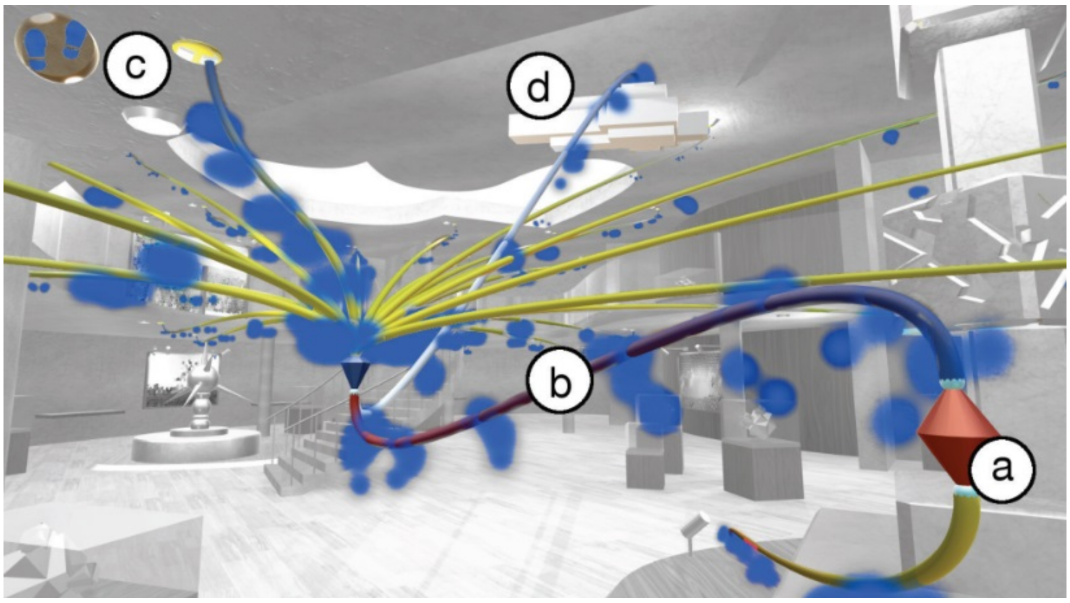


Figure 7.7: Ivy is an immersive visual programming tool for authoring IoT programs and visualizing sensor data. Users can (a) create program constructs, (b) establish logical links, (c) visualize data flows from real-world sensor data and (d) upload data to the cloud. Virtual port nodes (c) act as interfaces to real-world sensors, such as foot traffic readings in this museum scenario. Reprint from [39].

7.2 PROPOSED APPROACH

When designing a novel user interface concept for robot programming, we have identified the following issues of current solutions:

1) Mental mapping of robot instructions to the physical place in the environment: By the nature of robotics programs, the instructions in such programs are heavily linked to the environment surrounding the robot. Suppose we neglect the instructions related to the logical flow of the program (loops, branches). In that case, most robotic instructions deal with spatial information, such as where to move the robotic arm or pick an object. Lack of visualization of these spatial information leads to reduced program comprehension.

2) Context switching between programming device (e.g., computer) and the workspace: Most robot programming tools require a certain degree of constant context switching between the programming device, i.e., computer or teach pendant and the robot's workspace. This switching may negatively influence the performance of the programmer.

3) Low abstraction of the robot instructions, relations between the instructions, conditions, and parallel execution: The typical robot instructions operate on a low level of abstraction, dealing with setting the individual robot joints, manipulating digital outputs, or moving the end-effector to a specified point in space. While this level of abstraction is suitable for code programming, it is too low for visual programming, making the program chatty and hard to read.

Process-based vs. Object-based approach

The main goal of the programmer is to prepare a list of steps that describe: in what order the robot should perform various actions, using what objects should perform the action, and how and under what conditions should perform the action. The result is a sequence of actions – a program. In principle, this programming task can be implemented in two ways.

The first one is the so-called Process-based method. This programming method takes advantage of the top-down approach. The whole process is described using its inputs and outputs and followed by dividing the process into several subprocesses until it gets to the low-level problems. On the other hand, the object-based method describes the functionality of different low-level objects and allows to use of them to build a working system piece by piece. This method is also known as a bottom-up approach.

We used the real-world metaphor; when we describe the manufacturing process, we usually:

- First, we describe the environment: components, devices, tools, and objects that are in the scene and what they do or how to use them for the task,
- Then, we begin to describe the process step by step, including the links between the environment objects, their specific settings, and the expected outputs,
- Finally, we summarize the expected outputs and risk parts.

Based on this observation, we have decided to follow an object-based approach proposing a concept using spatially-aware AR on mobile devices.

Program representation

The three-dimensional Flowcharts inspired the program representation in our concept. Discrete operations (e.g. *pick the object* or *execute an operation*) are represented by nodes. As was already stated, most of the basic robotic task's operations are related to a particular place in 3D space, either by relation to a real or virtual object or directly to an absolute position in the scene. In our program representation, each node is spatially adjacent to the position where the action takes place. For example, a node representing the operation *Place object to the box* is located above the

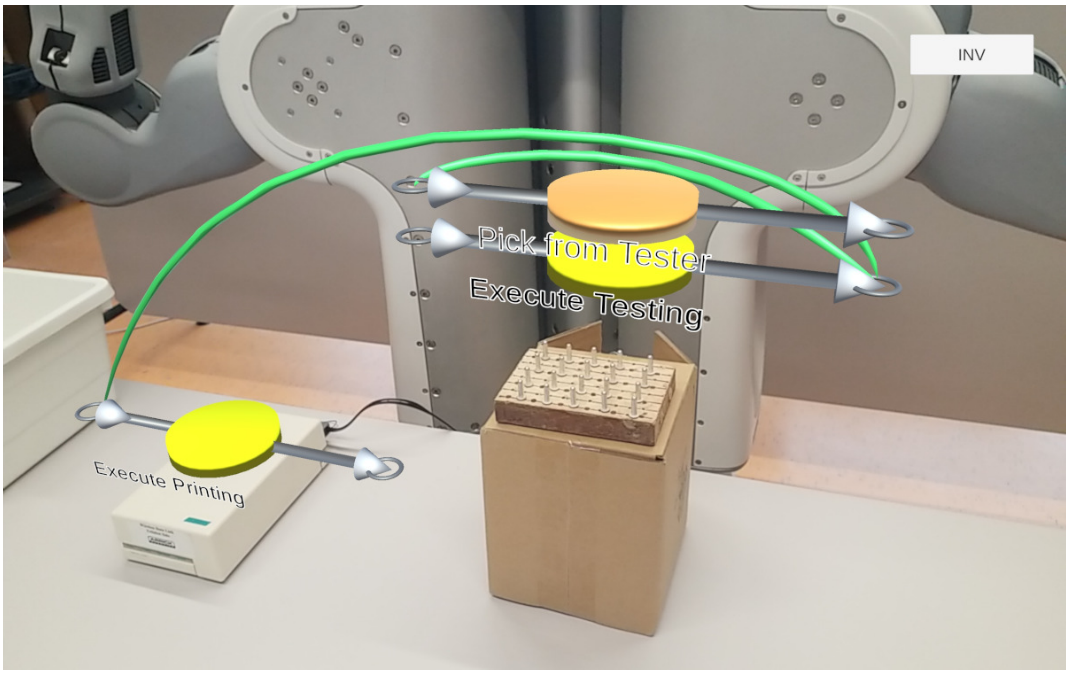


Figure 7.8: Parallel execution of the program. The *Execute testing* node is connected with *Pick from tester* node and *Execute printing* node. All of the *nodes* have set the same workpiece so that both paths will be executed at once during the runtime.

intended box. This adjacency helps the user mentally map the instructions to the physical space. Nodes hold information needed for their execution (e.g., the type of the object to be manipulated or position on the table where the object should be placed). In addition to setting individual operation parameters, linking these nodes to create a program flow is a key challenge for a usable user interface.

Each node has inputs and outputs. By connecting the inputs and outputs of various nodes, the user can define the flow of the program, i.e., the execution order of the actions. By connecting one output to inputs of multiple nodes, the user can specify conditional transition or parallel execution. The actual executed path is derived based on the parameters of connected nodes. In Fig. 7.8, parallel execution of the program is defined. Workpieces of all the nodes are the same, i.e., once the *Execute testing* operations are done, both *Execute printing* and *Pick from tester* operations will be executed in parallel. This approach is valid only when these parallel operations do not physically manipulate the workpiece. In this example, the robot will pick the workpiece using the *Pick from tester* operation, and the corresponding label will be printed simultaneously. This label will be stuck to the workpiece later in the program.

Conditional execution can be seen in Fig. 7.9. The *Pick from the table* node has set two different workpieces, e.g., red ball and green cube, meaning one of them will be picked. From this step on, two different *Place* operations can occur, each with

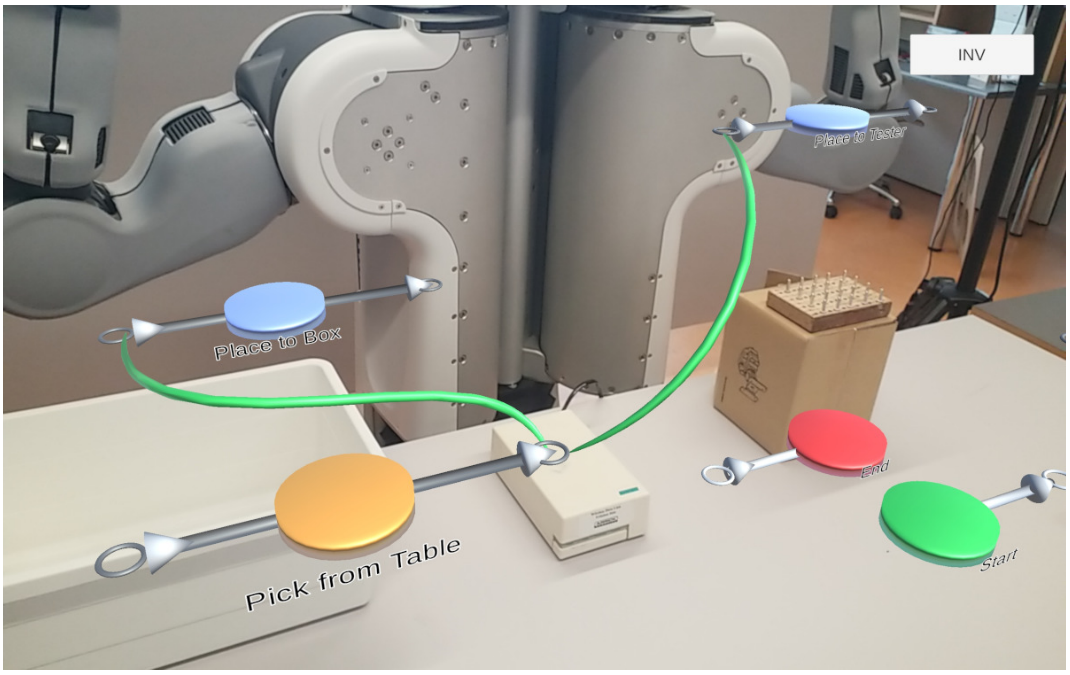


Figure 7.9: Conditional execution of the program. There can be seen *Pick from Table* node, connected with two *Place* nodes. The left and right *nodes* have set different workpieces. The flow of the program is decided during the runtime, based on the workpiece type picked in the *Pick from Table* node.

one of the aforementioned objects set as a workpiece. Based on the picked object from step *Pick from the table*, the corresponding *Place* operation is selected.

Spatially situated programming in AR

Spatially situated programming takes advantage of the spatial nature of robotic tasks. It is helpful in scenarios where spatial context is essential, like robots manipulating workpieces, picking them from the conveyor belts, or putting them inside the pressing machine. We propose a system that combines spatially situated programming with the knowledge of semantic properties of the objects in the environment: the knowledge that some object can be picked up or that a box offers to insert some other object. The user can benefit from that shared knowledge of the environment. Using this information, the user can define desired actions more effectively. The visual elements of the system are presented to the operator using AR, either in the head-up display or using a hand-held mobile device.

To evaluate the proposed approach, we have developed the user interface prototype using a hand-held mobile device. In cooperation with our industrial partner, we have selected a specific industrial use case, briefly described in the previous part. A detailed description follows.

Use case

The selected use case represents the process of electronic testing of the PCB. The PCB has to be inserted into the testing device (a.k.a. tester) and, based on the test result, either disposed of or forward to the next stage of processing. Besides, the corresponding label will be printed and pasted to functional and nonfunctional PCBs. A mockup of the testing facility was prepared and can be seen in Fig. 7.10. The mockup environment consists of the table with the PCB, the testing device, the printer, and the box for nonfunctional PCBs. Next to the main table is the other table intended for functional PCBs. The PR2 robot was placed behind the table to improve the feeling of the near future robotic facility. The whole procedure of the use case looks like this:

1. Pick the PCB from the table
2. Place the PCB inside the tester device
3. Execute testing
4. Do in parallel ...
    a) Pick the PCB from the tester device
    b) Print corresponding label
5. Place the PCB on the table
6. Stick the label to the PCB
7. Pick the PCB
8. Place the PCB to ...
    a) the box OR
    b) the other table

Step 4 represents the parallel execution of two operations simultaneously, as the robot picks the PCB from the tester device and the printer prints the label simultaneously. Step 8 represents conditional transition, as the PCB is placed either in the box or on the other table based on the result of the testing process.

7.3 PROTOTYPE OF THE USER INTERFACE

The prototype of the user interface was created using the Unity3D game engine. The ARCore[3] framework was utilized, together with the ARFoundation frame-

[3] https://developers.google.com/ar

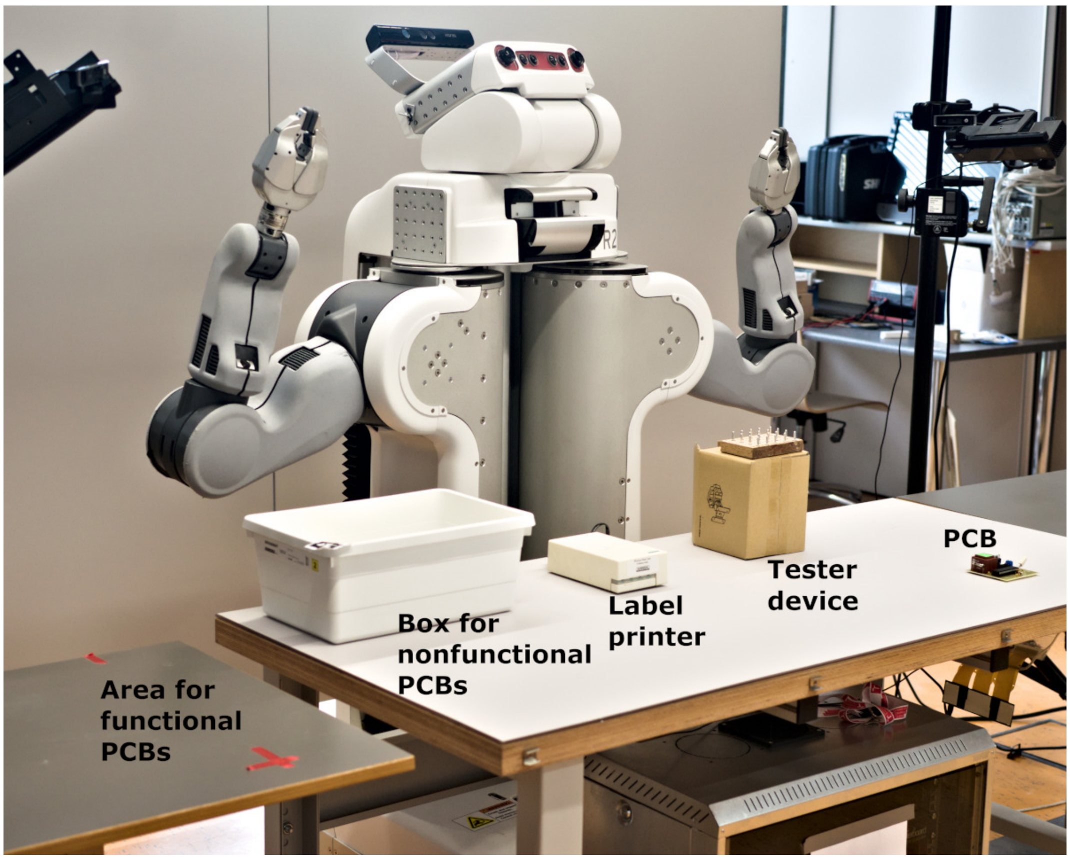


Figure 7.10: Testbed used during the experiment. On the table, from right to left, an example PCB, a mockup of the tester device, a printer of the labels, a box for disposing of nonfunctional PCBs, and another table for functional PCBs.

work, to register the mobile device's motion and track its position in the real world. The prototype was specifically developed for an Android-driven hand-held mobile device, the Samsung Galaxy Tab S4. The display shows the video stream from the back-facing camera with the superimposed user interface. The prototype is a single-purpose mockup, which allows no connection with the robot or other devices in the scene.

The virtual scene was created (see Fig. 7.11), Using Unity3D. It is spatially identical in scale and position to the real scene described above. The virtual and real scenes are mutually calibrated using the AR marker placed in the lower-left corner of the table. This calibration must be done once the application starts and uses the implicit AR-tag detection provided by the ARCore framework. The system simulates knowledge of the environment and context of all objects and devices on the table. Invisible virtual bounding boxes were placed around each physical object on the table to make the objects interactive for the user. These boxes serve as colliders to detect intersections with virtual rays, cast when the user touches the screen using their finger. This way, the users can interact with them by touching their projection on the screen.

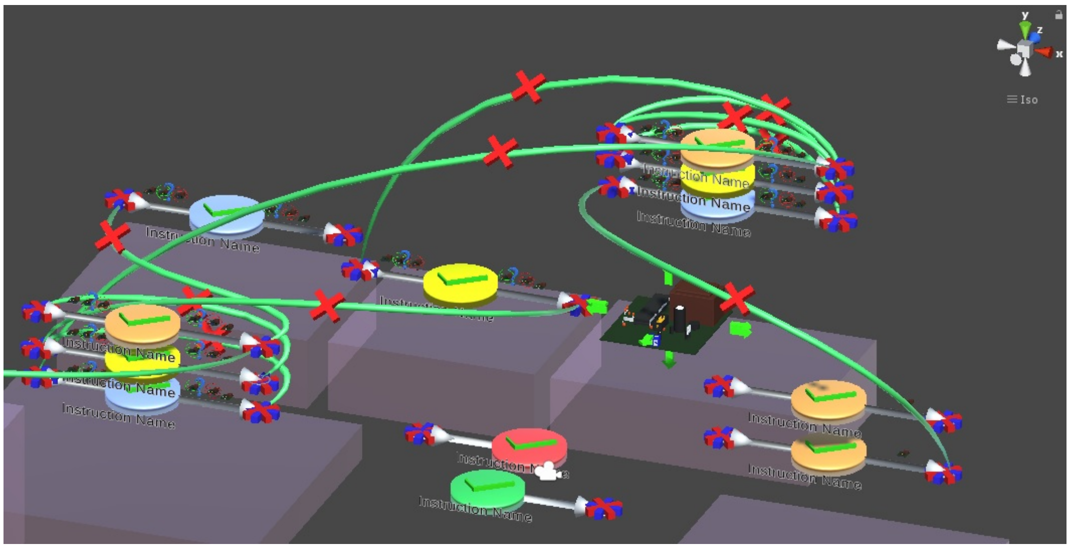


Figure 7.11: Unity scene of the prototype UI. Semitransparent boxes define interactive places, which users can use to define the intended operation. Above each of these boxes, there are *pucks* of various colors, representing different operations. They are connected with green splines, representing the program's flow (for clarity, only a subset of possible connections are displayed in this figure).

The mobile device's screen visualizes the programming process and serves as a main input for the operator. The application knows the position and semantic information of all objects in the scene (hard-coded for the prototype). Using the ARCore framework, the mobile device knows its position and orientation in the space, which enables the operator to interact with real objects by clicking on the 2D projection on the screen.

Several UI elements were designed for the prototype to allow users to interact with the system. These elements are both 2D and 3D. In this prototype, all elements representing different operations and their connections are static and prepared for the selected use case, as seen in Fig. 7.11.

Operations

In our prototype, each operation is represented by a so-called *puck* (see Fig. 7.12). The *puck* consists of a central disc, two circles representing the input and the output, and two pipes connecting the input/output with the disc. The input is placed on the left side of the *puck* (with inside the *puck* aiming arrowhead). The output is placed on the right side of the *puck* (with the arrowhead aiming outside of the *puck*).

The *puck* serves as a visualization of the operation and its parameters. At the same time, it provides the main input point for the operator. To change any operation's parameter, the user has to select the desired operation first. The so-called

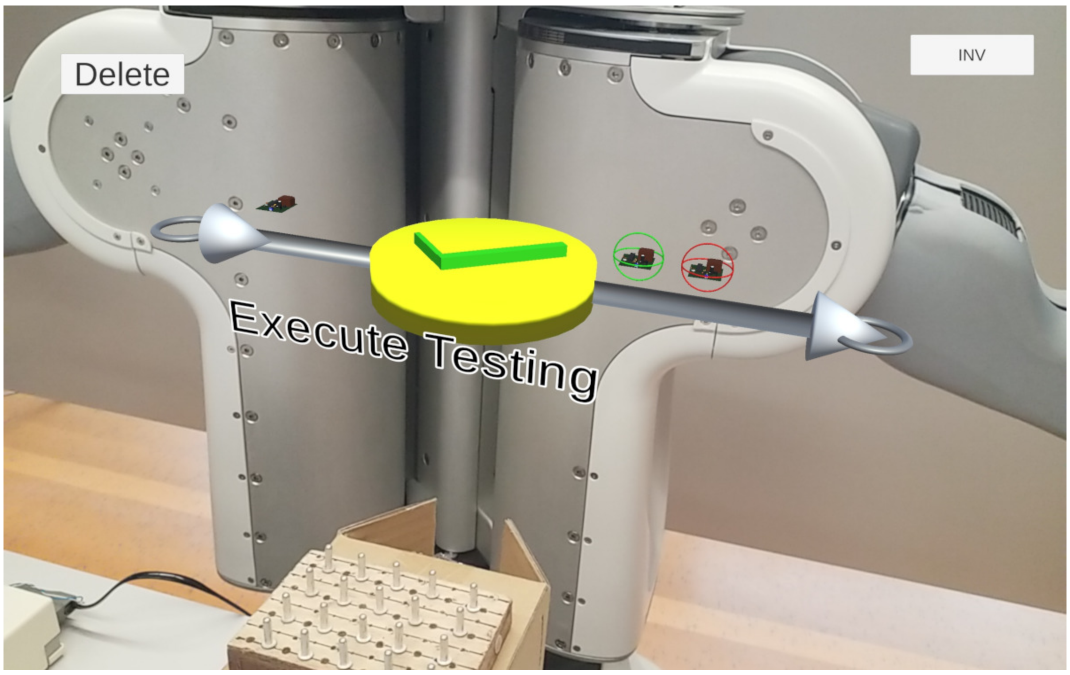


Figure 7.12: The *puck* consists of a central disc, two circles representing the input and the output, and two pipes connecting input/output with the disc. In the prototype, the type of the *puck* is represented by its color and the text placed in front of the disk. Above the pipes, small 3D models of input and output workpieces are placed.

edit mode was designed to enable this. Users can switch between the standard and edit mode by clicking on the screen's virtual projection of the *puck*. While in edit mode, only the edited and directly connected *pucks* are visible to the user, and the others are hidden to lower the visual clutter. In the edit mode, the parameters of the operation are visible.

Most operations only manipulate the workpiece without changing it, i.e., the workpiece on the input is the same as the workpiece on the output of the *puck*. Our use case has two exceptions: *Pick from the table* and *Execute testing*. The former has no workpiece on the input because it is the first operation in our program; therefore, the robot did not manipulate any object yet. The picked object is automatically set as a workpiece for the output and added to the inventory (which will be discussed later). The *Execute testing* operation works as follows. When the PCB is set as an input workpiece, it automatically creates two new workpiece types: *PCB_OK* and *PCB_NOK*. The former means tested and OK (functional), and the latter means tested and not OK (nonfunctional). These two workpiece types are also added to the virtual inventory.

Connections

The *pucks* themselves are not sufficient to define the flow of the program, as they only define operations but not the order in which they shall be executed. To define the flow, the operator can create connections between the *pucks* by connecting the output of one *puck* and the input of another *puck*. A green spline between these two *pucks* represents the connection. To make it easier for the user, once he clicks on the output of one *puck*, a big blue plus appears on the input of all other *pucks* and vice versa. By clicking on this plus, the connection is created.

In case of an incorrectly created connection, the operator can use a big red cross to remove the said connection. This cross is visible only for connections adjacent to the currently edited *puck*. Several connections can be attached to one output or input, allowing the user to define conditions and parallel execution.

Interactive objects and context menus

To define an operation for any physical object on the table (e.g., printer or tester), the operator has to create an appropriate *puck* above the object. The system benefits from the semantic information about objects in the scene; therefore, a context menu with every possible operation for the objects is generated in advance. This menu emerges by clicking on any object, allowing the user to define the desired operation. This was enabled by creating a clickable invisible bounding box around each object in the virtual scene (semi-transparent boxes in Fig. 7.11).

Inventory and teleoperating UI

While the user composes the program, each workpiece he uses in the program (e.g., PCB which shall be picked) appears in the inventory list. By clicking on the workpiece image in the inventory while in edit mode of some *puck*, the user can set this object as a workpiece for the said puck.

The operation *Place to the tester* needs to specify the 3D position of the workpiece while placed inside the testing device. A teleoperating user interface is prepared, allowing the user to move with a 3D model of the workpiece. There are two different approaches to controlling the position of the desk. The user can adjust the position in a vertical or horizontal plane using two joysticks on both sides of the screen (see Fig. 7.13). The other way to set the position is by using the DRAG button. When pressed, the desk moves in the same direction and speed as the tablet, so the operator can drag the desk by physically moving the tablet (see Fig. 7.13).

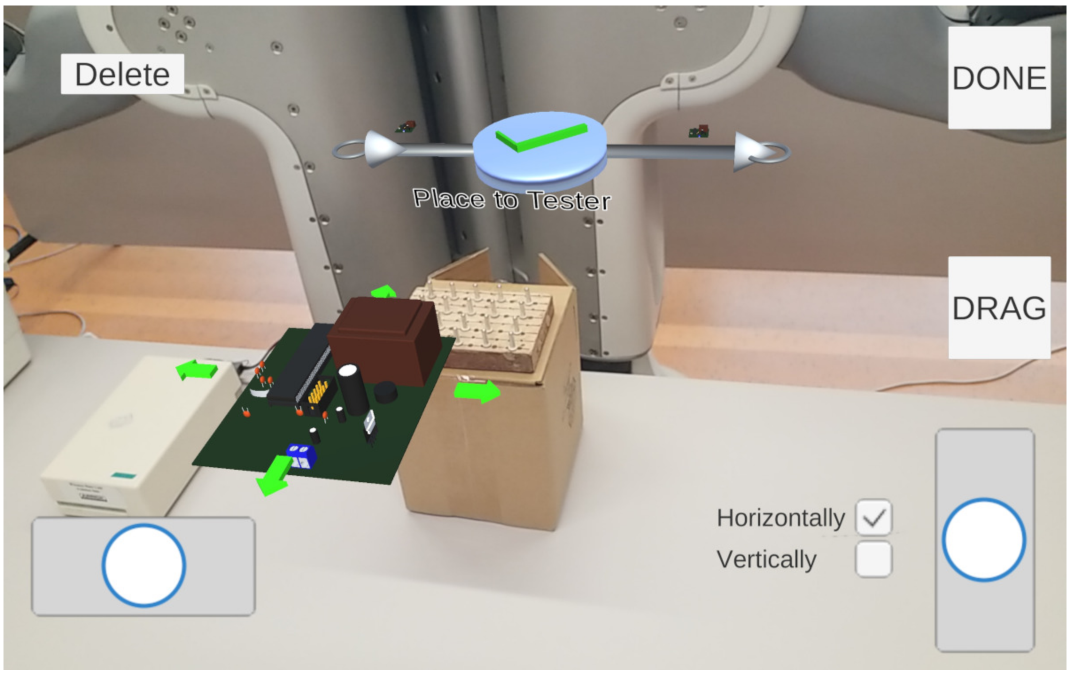


Figure 7.13: Teleoperating user interface for navigating the 3D model of the PCB to the tester device. There are two joysticks on the bottom left and bottom right side of the tablet. Next to the right joystick are two buttons for controlling whether the PCB should move in the horizontal or vertical plane. Above the right joystick is a *DRAG* button. When the user holds it, the 3D model moves in the same direction and speed as the tablet.

7.4 EVALUATION

This section provides qualitative results obtained from the user study with 7 participants. The prototype of the user interface (described above) using ARCore-enabled mobile devices has been developed to evaluate the proposed approach.

There were 7 participants of various ages and genders, with no or minimal programming knowledge and AR experience. These participants will be labeled as Participants A, B, C, D, E, F, and G. Table 7.1 shows the participants' demographic data.

Experimental Protocol

The experimental protocol consisted of 4 phases: orientation, training, programming, and discussion.

1) Orientation: As for the orientation, the moderator invited the participant to the testing site and introduced the evaluated system to him or her. The participant signed an informed consent form and fill out the demographic questionnaire.

Par.	Age	Gen.	Education	Experience with augmented reality	Experience with programming	Attitude towards new technology
A	24	F	bachelor degree	little	little	late majority
B	24	F	bachelor degree	some	little	early majority
C	34	M	master degree	none	none	early adopter
D	22	M	secondary	little	little	late majority
E	21	M	secondary	little	little	early adopter
F	24	F	bachelor degree	little	none	early adopter
G	33	M	secondary	little	little	early majority

Table 7.1: Demographic data of the participants. The scale for both experience-related questions was none, little, some, quite a lot, and many. The attitude towards the new technology scale is based on Rogers [116] diffusion of innovations.

2) Training: In the second phase, the participant was briefly introduced to the usage of mobile AR applications. The participant was told how to use the mobile device to create robot instructions (a.k.a. *pucks*), how to set the parameters of the instructions, and how to connect them to create the intended program. The participant had to complete several training tasks. In this phase, the moderator proactively helped the participant complete the tasks, answered all questions, and discussed visible misunderstandings with the participant.

3) Main task: The main task was presented to the participant by the moderator. They were asked to program the robot to pick the PCB from the right side of the table and place it on the testing device using the teleoperation menu. Once the PCB is inside the tester, the program should execute the testing process, print and stick the correct label based on the result of the testing process, and then place the PCB either in the box or on the other table (again, based on the result of the test).

After the task was presented to the participant, they began to work on it independently. The moderator was available to answer additional questions or help with problems with the prototype but did not proactively step into the programming process. Each participant worked until they claimed that the task was done. Moderator then reviewed the created program and either confirmed the correctness or suggested to the participant what they should alter.

4) Discussion: After completing the main task, the participant filled out the questionnaire. Besides, the moderator asked them their thoughts on the system and additional questions.

Sensors and collected data

The whole process of the experiment was recorded on several cameras. One was placed on the participant's forehead, aiming at the mobile device in the participant's hands. Another one was placed on the robot behind the table, aimed at the participant to see the whole scene and record the participant's movement. Two more cameras were recording the workspace from each side. The mobile device's screen was also recorded, with an indication of the participant's input. The participants and the moderator wore lavalier microphones to record their voices.

7.5 RESULTS AND FINDINGS

The section provides measured results and observed findings of the experiment. The experiment's main goal was to prove that non-expert users can program the selected use case using the ARCORO[4] system. We focused mainly on usability issues, the mental workload of the participants, and the overall user experience.

Qualitative and quantitative data

As a metric for the system usability, we chose the SUS [26] method as it can be used on small sample sizes with reliable results. It provides quick insight into the ease of use of the tested application. To evaluate the SUS score for our system, each participant had to score ten questions with one of five responses that ranged from Strongly Agree to Strongly disagree. An example of questions from the SUS questionnaire is: "I think that I would like to use this system frequently" or "I found the system very cumbersome to use. "

Table 7.2 shows the SUS score for each participant individually. The score varies from 0 to 100, where a higher number means a more usable system. The mean SUS score from all participants was 82.86, with a standard deviation equal to 9.29. According to Sauro-Lewis curved grading scale [123], the SUS score in the range of 80.8–84.0 is rated by grade **A** and is at the 90–95th percentile among other evaluated studies. This score shows promising potential for future research in this field and shows that the created prototype user interface is highly usable.

To measure the mental workload of the participants, a simplified NASA-TLX [54] method was utilized. The mental workload can negatively affect the operator's performance; therefore, measuring this attribute from the earliest prototyping phases is essential. Although the mental workload in laboratory scenarios cannot be gen-

[4] Augmented Reality COlaborative RObot

	SUS	TLX	UEQ ATT	UEQ PRA	UEQ HED	time to set (s)
A	95.00	25.00	2.67	2.50	2.12	535
B	80.00	25.00	2.00	2.42	0.75	427
C	67.50	47.22	1.17	2.00	1.75	460
D	85.00	27.78	1.67	2.25	2.25	507
E	92.50	27.78	2.67	2.75	2.88	431
F	82.50	19.44	2.00	2.08	2.38	521
G	77.50	19.44	1.33	1.83	0.88	806
mean	82.86	27.38	1.93	2.26	1.86	527
SD	9.29	9.41	0.58	0.28	0.72	130

Table 7.2: Detailed results of all measured metrics for each participant, together with mean and standard deviation (SD).

eralized directly to the workload in the real environment, it still can be helpful to reveal potential issues in the early stage of research. The NASA-TLX gives a number in the range from 0 to 100, where, contrary to the SUS score, the lower means better, i.e., lower number indicates a lower workload. The mean NASA-TLX in our experiment was 27.38 with a standard deviation of 9.41, which means that the workload was lower than in at least 80% of studies analyzed by Grier [49].

For an interactive system to be successful, a high-quality user experience is key. Among several methods to measure the user experience, we selected the UEQ [129] because of its simplicity for both participant and evaluator and reliable results. The UEQ scores from -3 to +3, where higher means better user experience. The system was overall rated as **Excellent** in all UEQ categories, i.e. *Attractiveness* (mean score 1.93, SD=0.58), *Pragmatic* attributes (mean score 2.26, SD=0.28), and *Hedonic* attributes (mean score 1.86, SD=0.72). All categories were evaluated using the standard UEQ benchmark [129].

The mean time for the main task completion was 527 seconds (SD=130s). The main task consisted of setting the following operations and their parameters and of creating connections between them: three times *pick an object*, four times *place object*, and three times *execute* (testing, printing, and sticking). For each operation, the workpiece had to be set using the inventory element (described above). For one of the *place object* operations, an exact position of the PCB inserted into the tester had to be set. The completion time excludes delays caused by prototype errors.

General findings

We found no fundamental problem during the experiment forcing us to reconsider the proposed approach. Although minor issues were observed or self-reported by participants, all participants were able to complete the task.

All participants reported that the *pucks* (representing operations) were unnecessarily large. In cases when there were more *pucks* above the same object, for instance, *place an object to the tester*, *execute testing* and *pick an object from the tester*, the state, and parameters of those *pucks* were unclear and it was hard to recognize mutual connections. The design of *pucks* needs to be refined, and a better strategy of *pucks* placement should be adopted in further versions to avoid it.

The participants were instructed to inform the moderator once they thought they had successfully finished the programming. Most of the created programs contained one or more errors, leading to failure during execution. Participant C explicitly reported that he cannot check if the program is correct. Participant A, in the end, went through all created *pucks* to check whether all parameters are correctly set and connections between *pucks* are as intended. After the moderator pointed out the errors, each participant was able to correct the error and successfully finish the task. This has shown that a debugging system has to be introduced, and better system state indicators should be involved. The program flow visualization needs to be altered to better support users' awareness of the program's correctness.

Only two of the participants found out that they could benefit from the active movement of the mobile device inside the scene to achieve higher accuracy when clicking on interface components. Most of them were standing at a certain distance from the table and using only vertical rotation in cases when the FOV[5] of the tablet was too narrow. Participant B stated that it was more comfortable for her to stand in one place to observe the whole situation and that she would appreciate the possibility of zooming the scene on the screen to avoid miss-clicks.

The usual procedure for most of the participants consisted of creating the *puck*, followed by creating the connection between the said *puck* and the previously created *puck*, repeated until the whole program was created. Participant A followed a different approach. At first, she created most of the *pucks* to label all desired operations, and once she was satisfied with *pucks*, she started to create connections between them. Participants A, B, C, and E used only one hand to control both joysticks (placed on different sides of the screen), while the rest used both hands, as was intended when designing the user interface. Participant A was the only one to use a DRAG button to set the initial position of the desk, followed by refining the final position using the joysticks.

[5] Field-of-View

Although minor issues were observed during the experiment, all participants rated the system positively. The participants agreed that the system is easy to use and requires no special knowledge from the operator.

7.6 CONCLUSIONS

This work aims to reflect current needs in programming robots for low and medium complex tasks in a shared collaborative environment. I have designed a new concept of robot programming using AR on a mobile device. The main goals pursued in the design of the new concept were:

- Eliminate the need to switch user context between desktop and work environment by mapping instructions directly into a real 3D environment.
- Reduce user mental stress by using semantic information about real objects.
- Increase the abstraction of instructions and their relations.

I have designed a method utilizing context-aware spatially situated programming to fulfill O2 and O3. It enables the user to define the spatial parameters and program flow simultaneously, using a combination of 2D and 3D widgets in the task space.

I have defined a simple use case inspired by the actual demands of the industry. In the experiment, we observed mainly the UI's usability, the user's workload, and user experience with the designed spatial programming concept. We have evaluated the user interface with seven non-programmers, which has shown that, despite some shortcomings discussed, this is the direction that can be taken. All participants were able to perform all the tasks independently after a short training. All participants evaluated the usability of the interface mostly positively.

Positive adoption of the new concept can also be attributed to the equipment that most users are used to working with. In the future, we want to verify this unambiguity and compare the usability of the concept with other yet less common devices, such as HoloLens glasses. In the following research, we will also focus on improving the orientation in the programmed task, solving the UX deficiencies observed in this study, and integrating the UI into a real robotic system.

8 SPATIALLY ANCHORED ACTIONS: COMPREHENSIBLE END-USER ROBOT PROGRAMMING IN AUGMENTED REALITY

The previously described concept of spatially situated robot programming using the mobile AR has shown strong potential in end-user robot programming. I have decided to investigate the possibilities of such an interface further. The initially evaluated prototype was focused mainly on defining the program flow, while the comprehension of newly seen programs was not evaluated. The robotic program is typically represented as a sequence of actions of just a few types, e.g., move line instruction, move joints instruction, or end-effector control instruction. These could be visualized in the form of diagrams or just as lines of code. By just looking at the program representation, it is hard to match individual action to the actual step in the program, i.e.: *"Which of the 20* `MOVE` *instructions is the one I am looking for?"*.

At the same time, the settings of precise spatial parameters have only been marginally explored, although it is a crucial part of robot programming. The precise spatial parameters definition is often quite challenging because the robot usually works with its coordinate system, which is not always aligned with the world and not apparent by just observing the robot. Many programming tools, especially those working with PbD, use the robotic arm to define spatial coordinates. The operator could use a direct manipulation with the arm (if the robot supports it) and manually drag the arm into the desired position. It is pretty fast, but the precision could be limited, and it could be tiresome for the operator's hands. The other possibility is to use the robot's teach pendant manipulation methods, such as jogging or joint manipulation. It would relieve the human's arms, and the precision is virtually unlimited. On the other hand, using this method could be slow. The third possibility is the combination of both approaches.

In the following research, I have focused mainly on robot program comprehension to further support the (O2) and easy and precise spatial parameters definition to comply with the (O3). I address the challenges mentioned above in the following chapter and propose an AR-based method for environment annotation and end-user robot programming. This method provides in-situ creation and visualization of the program and interaction with virtual and real elements of the scene. The method benefits from the semantic knowledge of the environment, i.e., of the presented objects and their arrangement. Only relevant actions are available for the programmer based on the present objects. The pose and shape of present objects could be used for the program's spatial parameters settings. Therefore, the environment must be annotated (i.e., objects and their poses specified) prior to the programming. The following chapter is based on the paper [70] which is submitted for review in the Virtual Reality journal at the moment of writing this thesis.

8.1 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 [2], Fanuc's Karel, or Universal Robots' URScript [113]. Although these languages offer relatively simple syntax and programming commands, they still require programmers with expertise in programming and robotics [7].

A certain form of simplified programming is offered by some teach pendants. However, they often possess high mental and physical demands, lack the ability to use common syntax structures, and have no option for visualization [6]; therefore, their usability seems to be rather low [125]. Offline programming tools, such as ABB RobotStudio[1] [36], 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, still requires extensive training. 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 [150].

Many 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 [46, 62, 94, 104], programming by demonstration (PbD) [9], tangible programming [130, 131], natural language interface [43], and some explored programming directly in the robot's space using AR [21, 91, 100, 102, 111, 158, 156]. The published works usually differ in the type of device used for the interaction and the level of robot programming. Some of them used a head-mounted display (HMD) to program the robot by setting trajectory waypoints [102, 111, 158]; others used projected spatial AR [91], visual programming in combination with visualization of spatial waypoints in the workplace [156], or an HMD in combination with a handheld pointer [100]. Apart from robot programming, AR has been found useful for visualizations of robot pro-

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

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

[3] robodk.com

grams and motions [119], inspection and maintenance [40], or training [15, 151]. 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 [138].

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 [82, 139, 142], and both have their strengths and weaknesses [98]. However, with their current implementation, they are usable for simple, small-scale environments [41] and non-complicated use-cases only, as hologram drifting can often rise to above 30 cm in challenging scenarios [124]. The use of these AR frameworks is 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 used in combination with another technology, e.g., kinesthetic teaching.

8.2 spatial programming paradigm and ui

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 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. For non-expert 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 immersion of the visualization (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

[4] developers.google.com/ar

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

spatial meaning, the programmer needs to merge these two pieces of information mentally.

In the case of robotic programs, not even the source code could provide insight into the program's logic. 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. The code could be tough to read and understand without properly naming methods or thorough comments. Some simulations using the 3D model of the environment could be utilized to overcome this problem. However, it suffers from the same challenges which had been already discussed, and preparation of such a simulation environment could be costly and time demanding.

The following section presents the conceptual paradigm of SAA, which utilizes specific 3D elements to visualize spatial information in the task space for development and program execution. These 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 sequence of the actions in the task space defines the robotic program. The paradigm deals with the effective usage of AR for visualization and interaction with virtual objects in task space. To allow fluent interaction with the virtual scene in the AR, the paradigm offers several interaction modes, which frees the user's field of view by presenting only interactive tools necessary for the current task. The paradigm defines two methods for virtual object manipulation: direct and indirect. Combining these two methods, fast and precise manipulation with objects is achieved in AR. Robot programming is a suitable scenario for the proposed paradigm, and we use it to explain and test the paradigm.

8.2.1 *Basic concepts*

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. 8.1). 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 said cube. This concept combines the spatial meaning of programmed action with its spatial parameters, which is crucial for robotic

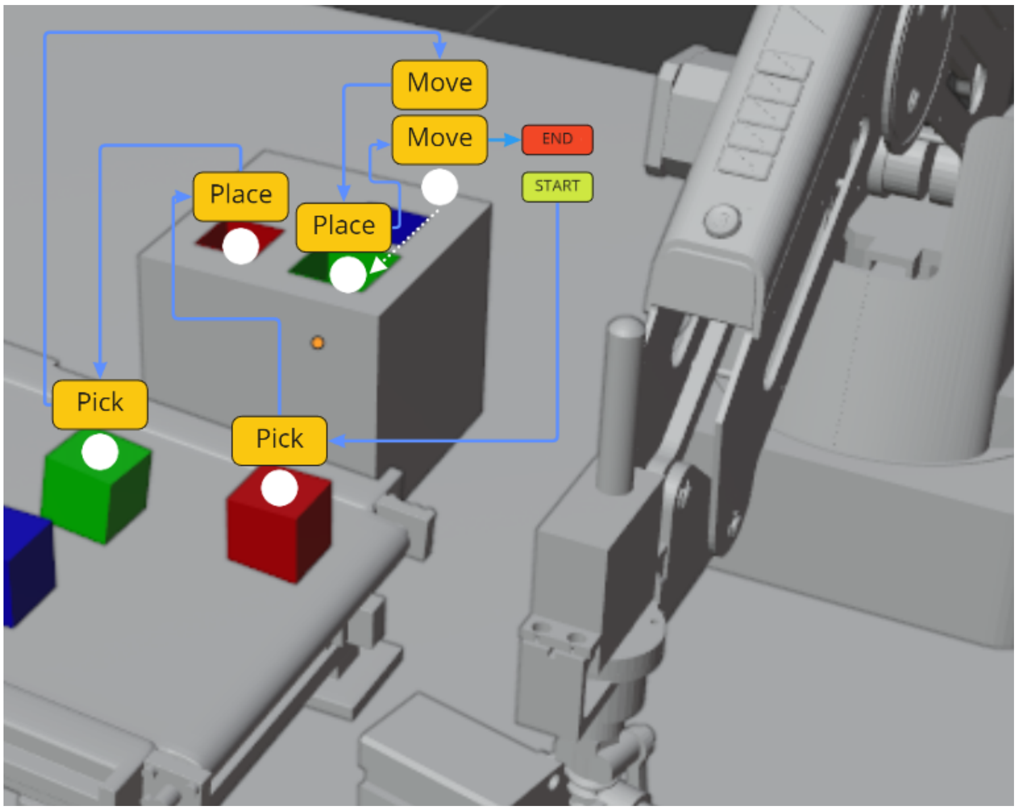


Figure 8.1: The visualization of the SAA concept. The white circles denote the *spatial anchors*, which serve for both the definition and visualization of spatial information. Above each *spatial anchor* is located one or more *actions*, represented by the yellow rectangle. The individual *actions* are connected by the blue lines, defining the program flow. Two *anchors* are connected by the white dotted line representing that the upper *anchor* is positioned relatively to the lower *anchor*.

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 so called *scene objects*, which are virtual counterparts of real objects in the scene. This enable the user to define some spatial parameter relatively to the real objects.

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

8.2.2 *Interaction modes*

To enable fluent interaction with minimal interface overhead, the proposed user interface introduces so-called Interaction modes. Based on the current interaction mode, only relevant tools are available for the user so 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 at a certain distance from the tablet in the forward direction 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 slightly 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*.

8.2.3 *Ergonomy of the user interface*

Most applications nowadays (including some AR/VR apps) use WIMP[6] 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, where the user observes the scene for some time, then looks at the head-up menu to interact, then looks into the scene again and so on. To avoid this, we followed

[6] Windows, Icons, Menus, Pointer

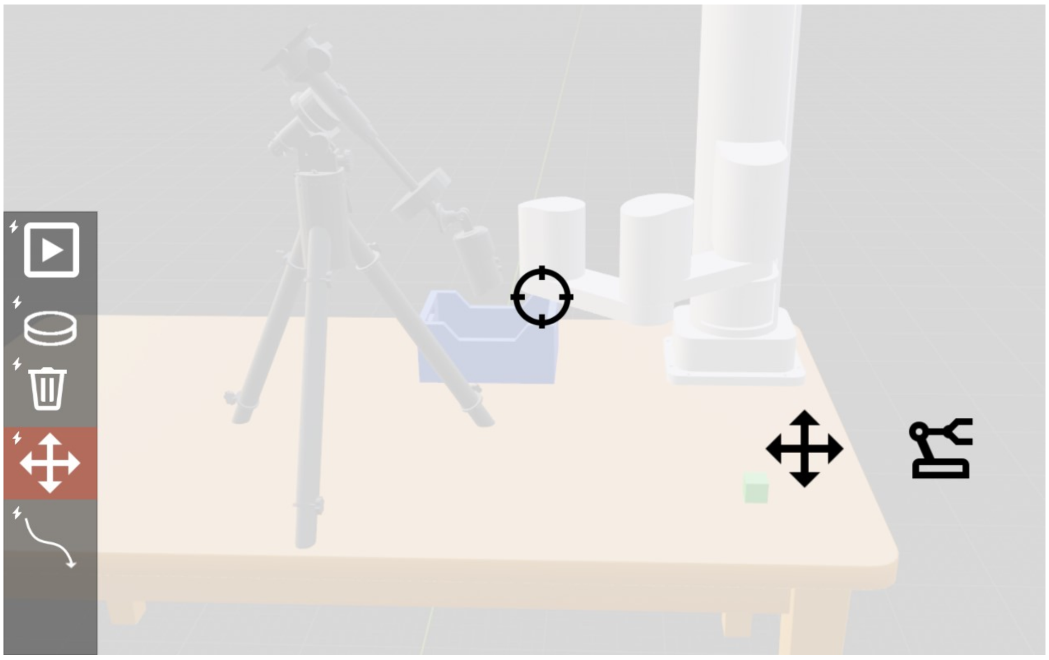


Figure 8.2: Schematic visualization of the user interface. The left side contains the main menu allowing the user to select the appropriate interaction mode. In the middle is a crosshair for indirect virtual object selection. On the right side are two context-aware fixed *mode buttons*, easily reachable by the user's thumb.

the design guidelines for UI elements in AR applications, as defined by the authors of ARCore framework[7]. 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 user to recover from missteps end errors by utilization of 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. 8.2. It consists of three parts. The left part contains the main menu, allowing the user 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 so-called *Mode Button*, whose appearance and function differ based on the currently selected interaction mode. The right

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

one serves to relax the robot joints in order to allow the operator to manipulate the robot arm. Both buttons are large enough and placed in the foreseeable place, according to the guidelines mentioned above.

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

8.2.4 *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 tens of centimeters [124]. On the other hand, when it comes to robot programming, there usually is a very precise device available for point specification – the robot itself. The robot could be used for the exact definition of points in space. 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 utilizes the robot's precision to specify certain places in the environment, which serve as **reference points**. Interaction widgets could be used to precisely define several **relative points** using the imprecise AR visualization using these reference points (see Fig. 8.1). The "parent" *anchor* is set using a precise method (i.e., manual guiding of robot or using computer vision techniques). Other *anchors* are set using a combination of 2D and 3D widgets with selectable precision (see Fig. 8.3). 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.

8.2.5 *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. 8.3, contains a palm-shaped button for direct manipulation – when pressed, the object moves with the device's movement, allowing fast movement over large distances.

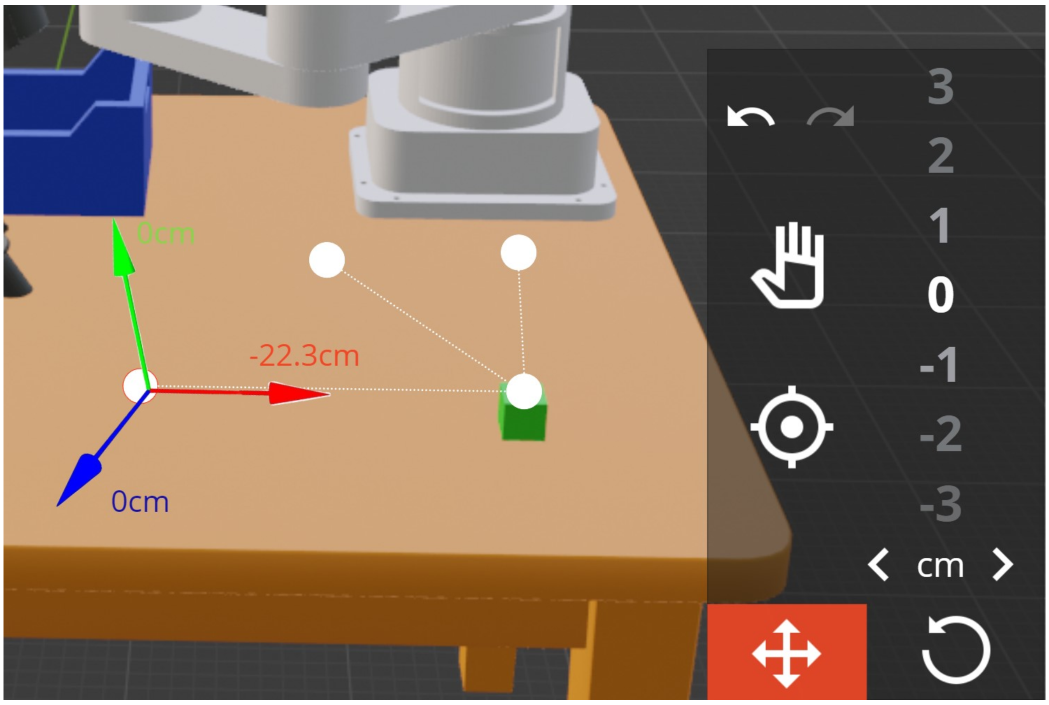


Figure 8.3: The schematic visualization of the tools available in the *transform mode*. The left side contains the 3D widget, so-called gizmo, rendered over the manipulated object. The right side contains the *transform menu*, with several interactive elements.

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. The numbers represent 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 to move the object by the exact length. 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 moved. We propose a 3D *gizmo* (see Fig. 8.3) 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.

In the left part of the *transform menu* are several buttons with an additional functionality. The arrows in the top serve for undo and redo operation. Bellow the palm-shaped button 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 of aforementioned **reference point** and subsequently define a **relative point** using the *rotary element*.

8.3 EXPERIMENTAL EVALUATION

The primary motivation for this work is to introduce a novel approach to end-user robot programming. 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 within-subject, with two conditions, where C_1 is the proposed prototype, and C_2 is a Blockly-based tool in the Dobot M1 Studio environment. We have stated four hypotheses related to the objectives above:

- H_1 – The user is faster acquainted with the program, seen for the first time, using the C_1 interface.
- H_2 – The C_1 interface is more usable than C_2 and puts less task load on the user.
- H_3 – The user can create a new program faster using the C_1 interface than the C_2 interface.
- H_4 – The C_1 interface provides similar precision for selected task as the C_2 interface.

The following chapter presents a user study we have prepared and conducted, which will help us to support or reject the stated hypotheses.

8.3.1 *Prototype*

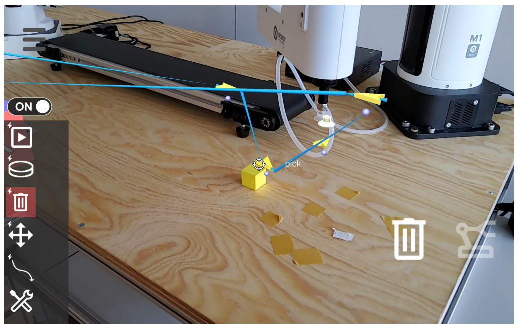
A functional prototype[8] was prepared for the experimental evaluation, containing basic functionalities for programming of pick & place-like tasks. The prototype application was developed in the Unity3D game engine, using the AR Foundation framework[9], which encapsulates the Google's ARCore[10], for 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-enabled application, following the guidelines described in the Section 8.2.3. The SAA (see Fig. 8.4) are visu-

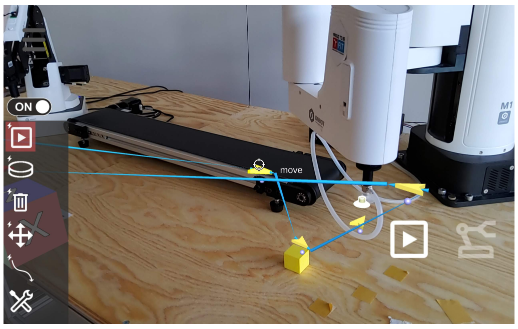
[8] Source code is available at github.com/robofit/arcor2_areditor.

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

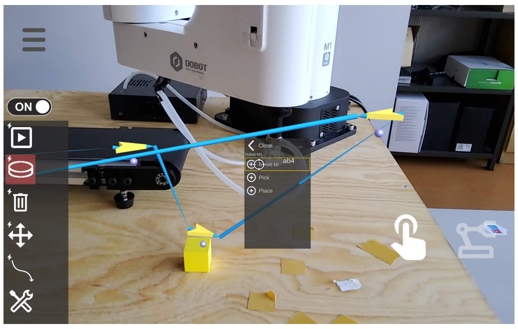
[10] developers.google.com/ar



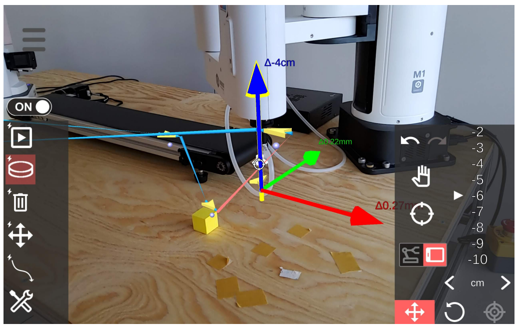
(a) The remove mode. The user could delete any virtual element by pressing the *mode button* when an object is selected.



(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.

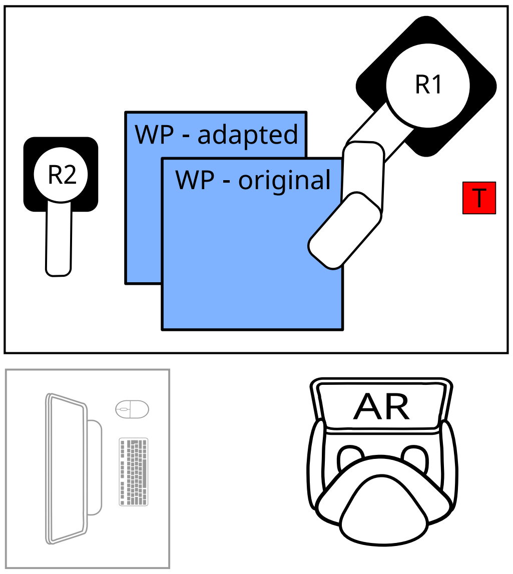
Figure 8.4: Graphical user interface of the prototype application. The left side contains the main menu for mode selection. On the right side is placed either *mode button* (a-c), depending on the selected mode, or the transform menu (d) in case the object is being moved. The central part serves for viewing the scene with the superimposed interface.

alized as yellow arrows located above blue spheres. The spheres represent spatial anchors, anchoring the actions for visualization and execution.

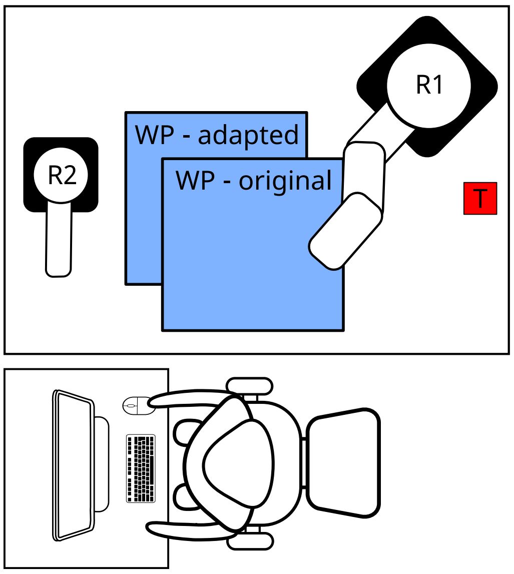
The prototype is fully functional, except for the object aiming procedure, which allows the user to set a precise object's position and orientation by navigating the robot's end-effector into several specific points on the object's body. In the experiment, this procedure was utilized to define the position of the workpiece. However, it was done using the WoZ approach for the sake of the experiment, which was unknown to the participants. Besides that, the participants interacted with a real, functional robot and created a robotic program from scratch.

8.3.2 *Experiment design*

The experiment was designed as a within-subject user study, comparing the two different interfaces – our prototype interface based on presented SAA (C_1) and the standard programming tool for the Dobot M1 robot – M1 Studio (C_2) with the Blockly tool. Both selected interfaces utilize visual programming and contain specialized elements for robot manipulation.



(a) Scheme for C_1. 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_2. 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.

Figure 8.5: 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 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.

C_1 utilizes a custom mobile 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. 8.5a).

C_2 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 equipped with a computer screen, mouse, and keyboard in front of the workplace (see Fig. 8.5b). 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.

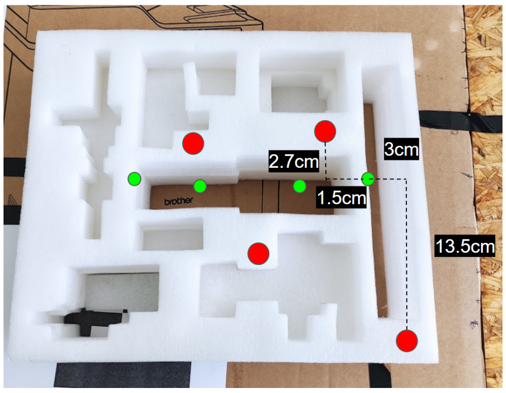
To minimize learning and transfer bias caused by the study being designed as a within-subject, the order of both conditions is randomized for each participant. For the safety purposes of both robot and subjects, each participant was thoughtfully instructed on how to control the robots safely, the maximal 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.

8.3.3 *Experiment protocol*

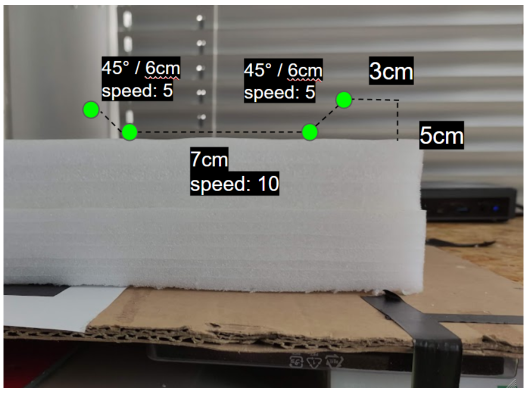
Each experimental run was organized as follows. At first, the moderator welcomed the participant and asked them to sign an informed consent and fill in a demographic questionnaire. After that, a brief workplace introduction took place.

The moderator randomly assigned the first condition to the participant and introduced them briefly the programming tool, and after that, the participant began with the training task (T_1). 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 releasing (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.

Following the training, the visualization task (T_2) took place. An existing program was presented to the participant. Their task was to identify some program steps according to the moderator's questions. The participant was explicitly in-



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



(b) The trajectory points (green circles) and their mutual positions.

Figure 8.6: The drawing of the intended trajectory for the main task, superimposed over the workpiece used for the experiment.

formed 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 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_1: 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.
4. the bigger robot to pick the box from the table.

Questions for C_2: 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.

Next, the main task (T_3) was presented to the participant. It simulates precise robotic manipulation with workpieces in a structured environment. Specifically, the robot should pick a cube and perform simulated grinding by following a specific trajectory defined by a technical scheme (see Fig. 8.6), which was available to the participant during the whole 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 put the cube back in the original spot on the table. The experiment task was the same for both conditions. For the C_1 condition, the participant had to annotate the position of the workpiece first as a part of the T_3 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. 8.6a) 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).

When the T_3 was successfully programmed by the participant, the moderator moved the simulated workpiece to the new place, and the participant had to adapt the program (T_4). In the case of C_1, 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_2, setting a new position for all the waypoints needed to be done again. For simplicity, the participants were told only to set the first waypoint.

During all the tasks, the participant was allowed to test the execution of both 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.

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.

8.3.4 *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. As a subjective metric, standard questionnaires were selected. Namely, the NASA-TLX [54] for measuring mental and physical load, and the SUS [26] 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_1 condition, the Handheld Augmented Reality Usability Score (HARUS) questionnaire [122], which is explicitly designed for the usability of handheld AR interfaces, was incorporated.

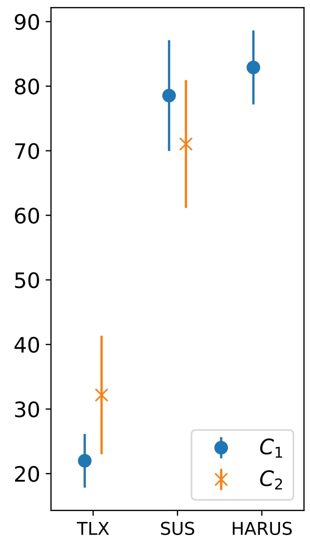
8.4 RESULTS

This section summarizes the user-study results and provides its analysis and interpretation. Regarding the task completion time measurement, intervals where participants asked questions, a technical problem occurred, or when the moderator had to intervene, were subtracted to measure a pure task completion time. 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.

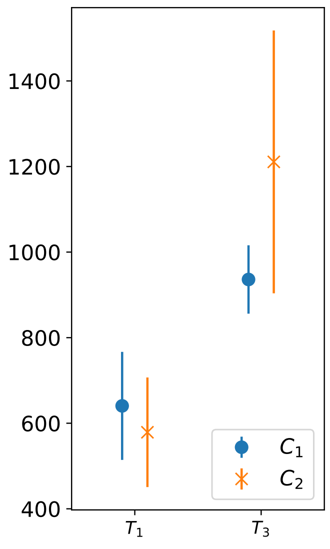
The user study was conducted with 12 subjects of various ages, self-reported genders, and technical backgrounds. Eleven participants identified themself as males; one identified themselves as female. Most subjects are ordinary shop-floor workers, students, or graduates from humanities colleges with little 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.

8.4.1 *Quantitative and Qualitative Data*

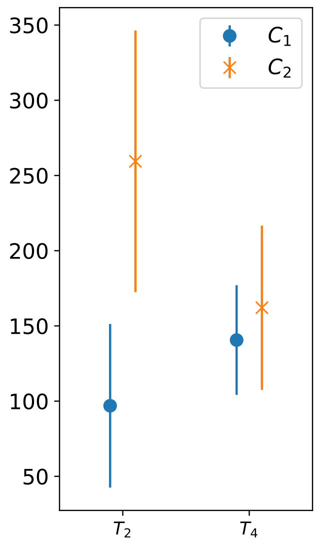
Results from SUS and NASA-TLX questionnaires (shown in Fig. 8.7a for both conditions) show that, on average, the participants perceived a lower task load using the C_1 interface and ranked it as more usable. The mean NASA-TLX score for the proposed SAA interface (C_1) was 21.99, which is less than 32.18 for the C_2. Regarding the usability of the interfaces for both conditions, the SUS questionnaire results show that participants consider the interface from C_1 more useful, scoring 78.54, compared to the C_2, scoring 71.04. However, differences are not significant for both metrics according to paired t-test ($p = 0.074$ for NASA-TLX, $p = 0.312$ for SUS); therefore, the H_2 can not be confirmed. Besides, the C_1 was scored 82.90 using the HARUS method, which is specifically designed to measure the usability of hand-held AR systems. The score is higher than that of comparable interface SlidAR [107], which is aimed at virtual object manipulation and scored 76.3 (SD=10.83).



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



(b) Time (in seconds) needed to complete T_1 (training) and T_3 (main) tasks.



(c) Time (in seconds) needed to complete the T_2 (visualization) and T_4 (adaptation) tasks.

Figure 8.7: Comparison of subjective and objective measurements (mean values and corresponding 95 % confidence intervals) for conditions C_1 (proposed method) and C_2 (standard method).

The training time (T_1) was comparable for both interfaces, although slightly longer with the C_1 interface (see Fig. 8.7b). Contrary, the main task (T_3) was significantly faster with the C_1 interface according to the Wilcoxon test ($p = 0.042$); therefore, the H_3 was confirmed.

In the adaptation phase, the users were told to:

- complete the aiming procedure for the workpiece in the new position for C_1 condition,
- set the position of the first point of the trajectory for C_2 condition.

The completion times in Fig. 8.7c show that the adaptation using the C_1 condition was significantly faster even when the participants did not adapt the whole trajectory in the C_2 condition, showing that the gap will get even wider with the increasing amount of points in the trajectory.

Analyzing the completion times for the T_2 (visualization task), it was shown that for the C_1 condition, the participants required significantly less time to answer the questions (see Fig. 8.7c). This suggests that the AR interface greatly supports the user in program comprehension, especially for the actions with the spatial information, which are crucial for robotic program understandability; therefore, the H_1 is confirmed. The discussion with the participants showed that they felt more certain when they had to identify the program steps using the SAA presented in AR. Most of them could identify each step quickly by just 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_1 condition), which has no clear spatial information. Most of the participants could identify it after a short time, which shows that the users can identify even actions without clear spatial information using the proposed interface. When using the M1 studio interface (C_2), most 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 highly successful but very time-consuming. The participants, on average, needed 1.17 attempts (SD: 0.38) to identify the correct action for the C_2 interface and 1.2 attempts (SD: 0.45) for the C_1 interface, but over a significantly longer period of time.

8.4.2 *Preferences*

According to the questionnaire of the C_1 interface, the vast majority, specifically 64.3 % of participants, preferred the rotary control element for the precise movement of virtual objects. Both setting using the robot manipulation and the free-form setting using the tablet motion were preferred by 16.67 % of participants. On the other hand, for setting the coordinates where the approximate position is sufficient, 50 % preferred using the robot manipulator, 33 % preferred the free-form setting using the tablet motion, and 16.7 % preferred the rotary control element.

According to the questionnaire regarding the C_1 interface, the participants considered the rotary control element more useful than the free-form movement of the virtual objects (see Fig. 8.8). We argue that it is primarily because of the selected task, as it required settings of several precise positions. In contrast, the setting of non-precise positions was unnecessary, and the participants utilized it only for a couple of intermediate movements.

The participants also liked the inserting of new spatial anchors on the current position of the robot's end-effector (see Fig. 8.8) than freely to the space (in front of the tablet). 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. As the participants had to set a path for the robot based on specification, they usually followed this pattern:

1. Set 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 8.2.5), which sets the position of the waypoint to another virtual object (previous waypoint in this case). Most of the participants struggled a bit on this at the beginning, and they would appreciate, according to our observations and discussion with them, the possibility of adding a new spatial anchor to an existing one, similar to adding it to the position of the robot's end effector.

Most users considered the robot motor's unlock button very useful. However, some did not like the dead-man-trigger concept, as they reported that it is hard to press that button while holding the tablet with one hand. Moreover, it was difficult for some of them to move the robot with one hand only.

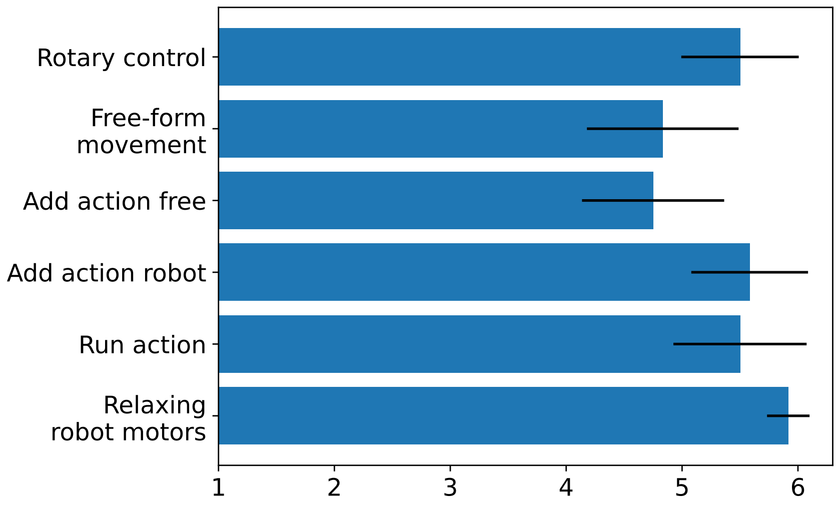


Figure 8.8: Usefulness of selected features of C_1 interface, rated by the participants on a scale from 1 (useless) to 6 (very useful).

8.4.3 *Observations*

The participants generally liked the possibility of quickly executing *actions*, using the C1 during the T3, as it enabled them to check the reachability of the spatial anchors. With the C2 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 C1 interface because they saw the position of the spatial anchor in the AR.

Several participants reported that at the beginning, they were stressed out by the C1 interface, mainly because of a rich set of functions, compared to the C2 interface. Moreover, they claimed that the 3D interface elements were entirely new to them, and it took some time to get used to them. Nevertheless, most of them agreed that after a short time, they got used to the controls and the programming was easier than with the C2 interface, despite their initial concerns.

For the task T2, all but one participant preferred the C1 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 C1. 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.

Only two participants preferred the programming using the C2 interface over the C1. Both are rather technically oriented people; 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 C2 interface. Both of them stated that the C2 interface was more straightforward for them and reminded them of the tools they are or were using at their jobs.

All participants struggled with the visualization and control of the gizmo element in C1. They were often unsure which axis was selected or accidentally selected the wrong one. Many participants struggled with the magnitude of the transform step selection, causing them to either move the object at the wrong length or wonder why the object is 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 of both the magnitude and direction of the desired movement.

Most participants considered the blue lines between the individual actions in the C1 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_2 the users can modify the coordinates in textual form with virtually unlimited precision. The C_1 preserves the possibility to set 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_4 to be confirmed.

8.5 CONCLUSIONS AND FUTURE WORK

This chapter presents a novel paradigm for spatial programming in AR on mobile devices. The paradigm defines SAA for program visualization, their manipulation in real 3D space, and UI elements and rules for interaction in AR on mobile devices.

The new concept was introduced and tested on a robot programming task. A fully-functional prototype was created using a tablet-like handheld device, which was evaluated with 12 potential users and compared to the existing visual programming method, as required by the O_4. 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 O_2. All participants successfully finished all tasks using both interfaces. The visualization (T_2) and main (T_3) tasks were done significantly faster using the proposed interface, Therefore, it was shown that the simplicity of program creation is at least similar to the standard tool and the users are able to create a new programs faster. The study also aimed to lower the users' task load no significant task load reduction was revealed. However, it was relatively low for both tested conditions. A higher number of participants could show significant differences. One of our objectives was to provide good ergonomics for the mobile AR interface. 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.

Moreover, we moved most of the interaction elements from the on-screen menus to the 3D scene, 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, according to the O_3. The users could finish the task with similar precision in both conditions.

In future work, we plan to investigate some drawbacks revealed by the study. The 3D gizmo widget for axis selection was sometimes unclear for the participants as they were unsure which axis was selected or what distance / angle magnitude was currently selected for transformation. 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 troubles with robot manipulation. Moreover, we will evaluate the feasibility of the proposed concept in different contexts than robot programming in SMEs such as home automation, where there is also high demand for end-user programming techniques and, at the same time, a need to set spatial parameters as, e.g., the definition of various kinds of zones.

9 IMPROVED INDIRECT VIRTUAL OBJECTS SELECTION METHODS FOR CLUTTERED AUGMENTED REALITY ENVIRONMENTS ON MOBILE DEVICES

The previous research has shown a strong potential for spatially situated robot programming using AR on mobile devices. An inevitable part of such an interface is the selection of virtual objects in cluttered environments. In the interface presented in Chapter 7, the selection was made using the direct method, where the users select the objects by touching their projection on the screen (see Fig. 9.2). Due to the used large-screen device, the users must hold the device with one hand, so the other hand was free for the selection, or they often had to change the holding position to reach the whole screen. It caused discomfort for the users. The interface proposed in the Chapter 8 utilized a simple indirect method. It was sufficient in a sparsely cluttered environment. However, in a more cluttered environment, the selection became more challenging. The users had to move with the tablet heavily to get a clear view of the virtual object they wanted to select. Therefore, I have decided to more investigate various interaction method in the following research in accordance with the O1. This chapter is based on the previously published paper [69]. It presents a preliminary study with three different methods for object selection.

Traditional approaches for object selection might be divided into two categories: direct and indirect. For computers, the most widespread method for object selection is indirect control of the graphical cursor, either by mouse, keyboard, or touchpad. The direct method is usually the most common for touchscreen devices – using either the user's fingers or a stylus.

When it comes to a 3D user interface, such as AR on mobile devices, Bowman et al. [25] state that the quality of interaction with 3D objects has a profound effect on the quality of the entire 3D user interface. They also state that selecting and manipulating virtual objects is one of the most crucial features of such an interface because "*. . . if the user cannot manipulate virtual objects effectively, many application-specific tasks simply cannot be performed*" [25].

In the case of cluttered scenes with partially or fully occluded objects, traditional methods may begin to lose their breath in terms of precision or speed [121, 157]. The problem worsens on mobile devices, where the primary selection tool is usually a human finger. In the case of large displays (typically with tablets), the ergonomics of the whole process need to be taken into account due to the weight of the device, especially when it comes to long-lasting operations, such as robot programming.

I have selected spatial visual programming in AR as a representative task, similar to the one presented in our previous work [67]. It contains a relatively high number of virtual objects inside a small area, partially occluded (depending on a view angle).

I have prepared an experiment to compare two indirect and one direct method for object selection in AR:

1. Combination of the crosshair and a head-up side menu containing a set of nearby objects.
2. Combination of the crosshair and in-space hierarchical menu.
3. Direct touch on the virtual object.

We conducted a pilot study ($n = 3$) to get a first impression of both developed methods and verify both experiment design and prototype application. This evaluation is a part of our ongoing research on simplified robot programming.

9.1 RELATED WORK

The problem of selecting objects on a screen is well studied. However, there are many specifics when it comes to AR on handheld devices. First, objects on the screen are not stable during interaction due to hand tremors or visual tracking instability. Also, techniques intended initially for VR, as Go-Go [108], for instance, are less usable on handheld devices [157].

In the literature, Fitt's Law [42], a technique to quantify the difficulty of a target selection task, is commonly used to compare selection methods. The original one-dimensional version was later adapted to 2D and even to 3D selection problems [32, 109, 141]. However, it implies several unrealistic limitations: participants must be seated and locate homogeneous un-occluded targets on a plane. There exist several attempts to create more realistic conditions, where targets were: displayed at mixed depth [109], of different sizes with variable levels of occlusion [10], highly occluded, and with similar appearance [95]. Although those publications made some aspects of evaluation more realistic, they were carried out on purely synthetic tasks in an empty environment. The evaluation in a realistic industrial environment exists as an isolated example [105].

There is also a lack of experiments carried out on larger than phone form factors (device's physical size and shape); there are just indications that indirect methods might be more suitable for them [112, 147]. Moreover, there are signs that indirect methods could be preferable for long-lasting tasks [121].

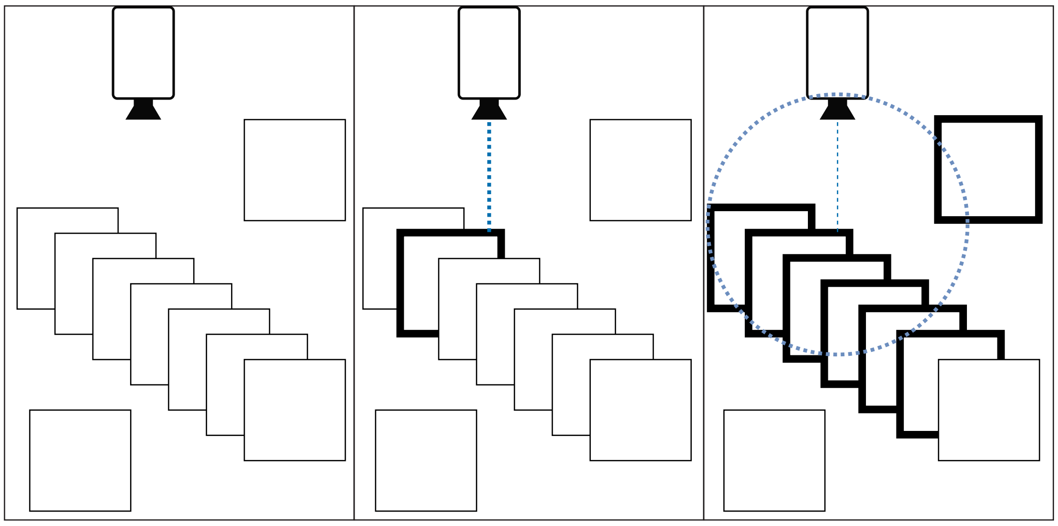


Figure 9.1: Principle of our cluster selection in two dimensions. Left: the camera observes a scene. Middle: a ray is cast from the camera's center and hits an object (square with thick borders). Right: in the position of the hit, a circle (sphere in 3D) is generated and all objects located inside or colliding with are included into the cluster (squares with thick borders).

An example of a selection technique specifically designed for handheld devices could be DrillSample [95], intended to provide accurate selection in dense AR environments, optimized for one-hand operation on the phone. It is a direct method, using ray-casting (touch) and an optional refinement step. A set of selection methods for phone-sized devices and dense AR environments that outperformed ray-casting and Go-Go was proposed and evaluated in [157]. A screen-centered crosshair was compared with a relative one, bound to the physical object's frame in [147] for both phone and tablet. The relative one was more accurate and less sensitive to the registration jitter and the device's form factor. The list-based selection, with icons displayed on the side of the screen for objects nearest to the crosshair, was compared with a touch-based selection in [121]. The list-based method was designed to minimize the number of touches by taking advantage of device motion and is recommended for crowded scenes to select multiple objects during longer-lasting tasks.

As throughout all the thesis, the main use case in this research is robot visual programming. Even a relatively simple pick and place task results in a dense AR environment, with many virtual objects of different appearance and semantic meaning, typically clustered nearby spatially important points. The task requires a large screen; therefore, we were interested in tablet-like devices. To allow long-term usage, we have developed methods enabling users to hold the device with both hands, control the interface using their thumbs, and evaluate them on a realistic task.

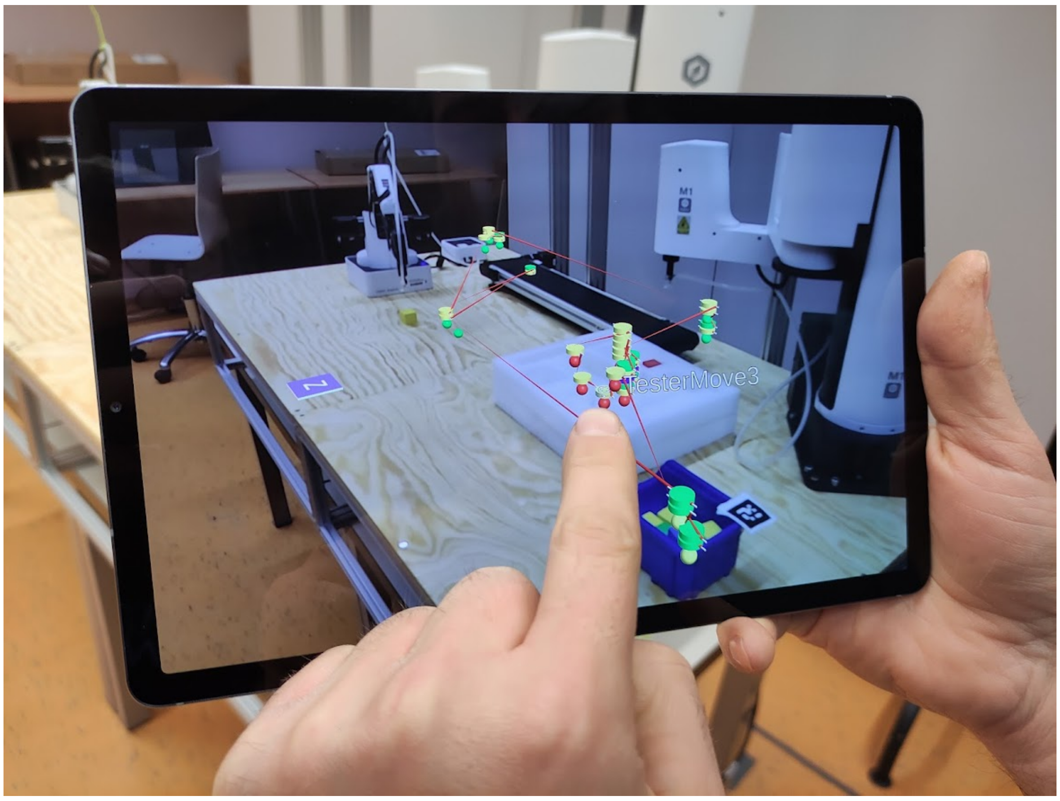


Figure 9.2: Direct touch method. The user selects the object by touching its visual representation on the screen.

9.2 PROPOSED METHODS

The necessity to select virtual objects in dense AR environments arose during the development of robot programming tools in AR using tablet devices where specific virtual objects represent spatial anchors, robot actions, and process flow. Such approaches usually have a solid connection to the natural environment and thus show strong potential in AR [33, 67]. Even relatively simple tasks usually involve many virtual objects in the scene with some occlusion. In such an environment, the selection becomes problematic, so there is a need for fast, accurate, and easy-to-use methods, specific for large screens.

We have proposed two indirect methods for precise virtual object selection in heavily cluttered environments in AR. Both of them work with spatially clustered objects and use the following algorithm to obtain the cluster from the scene:

1. Cast a thin ray from the center of the screen (indicated with the superimposed crosshair) and add the first hit object to the cluster (see Fig. 9.1, middle).

2. If the cluster is empty (i.e., the ray hits no object), cast a thick, square-shaped ray with a side of size X cm from the same origin, allowing to select even tiny objects, and add the first hit object to the cluster.

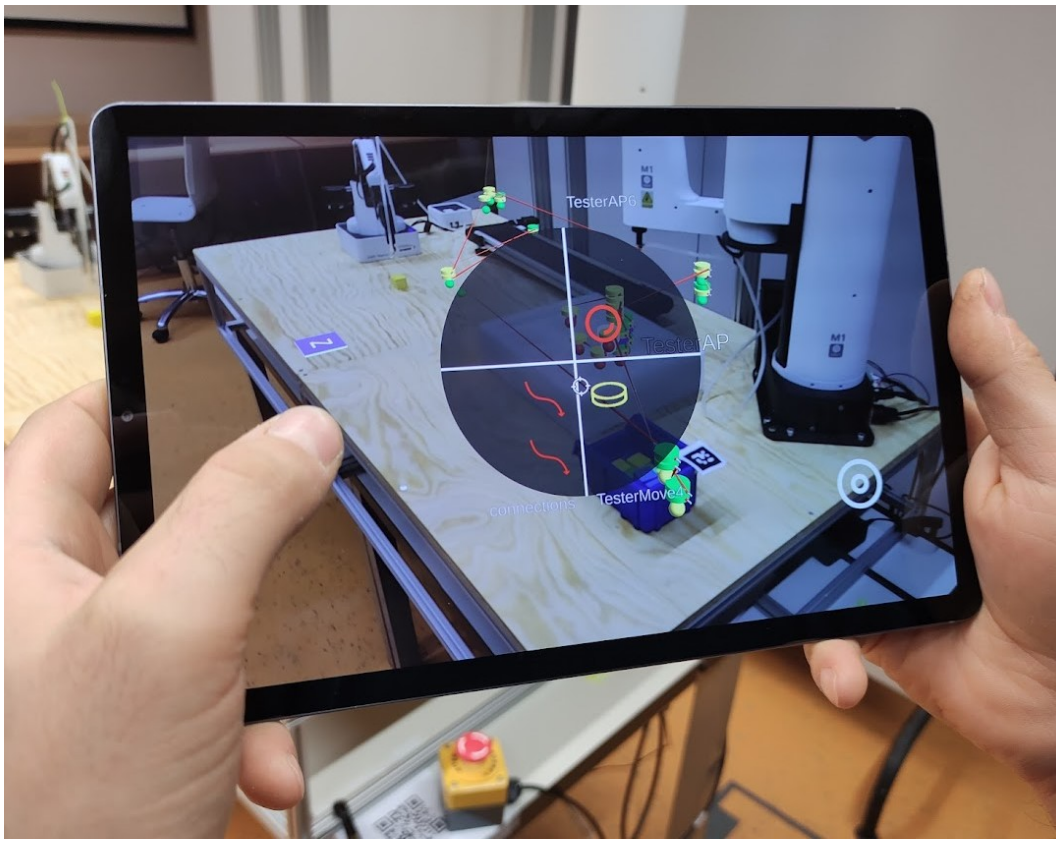


Figure 9.3: Spatial hierarchical menu. The user selects a cluster of objects by device movement, followed by selecting quadrants of the menu.

3. If the cluster is still empty (i.e., no ray hits any object), return an empty cluster.

4. In the position where the first object was hit by the cast ray, construct a virtual sphere with a radius of Y cm. Search for all objects colliding with the sphere (see Fig. 9.1, right). Add all these objects to the cluster and return it ordered by their distance to the first hit object, from nearest to farthest.

The size of the thick ray and the virtual sphere were selected empirically for our experiment. The X, i.e., the size of the thick ray, was set to 1 cm, and the Y, the sphere's radius, was set to 3 cm. These values depend on the nature of the AR application, the number of virtual objects in the scene, and their size. The problem of automatically tuning these parameters is out of the scope of this thesis and is not discussed here.

Proposed methods are meant to help the user select spatially clustered objects in a cluttered environment. Each method provides different access to hard-to-reach or occluded objects by either hierarchical or flat representation. One of the proposed methods uses the head-up menu, and the other uses the in-space menu, which allows us to observe whether the attention switches between the scene and the head-up menu will be problematic or annoying for users.

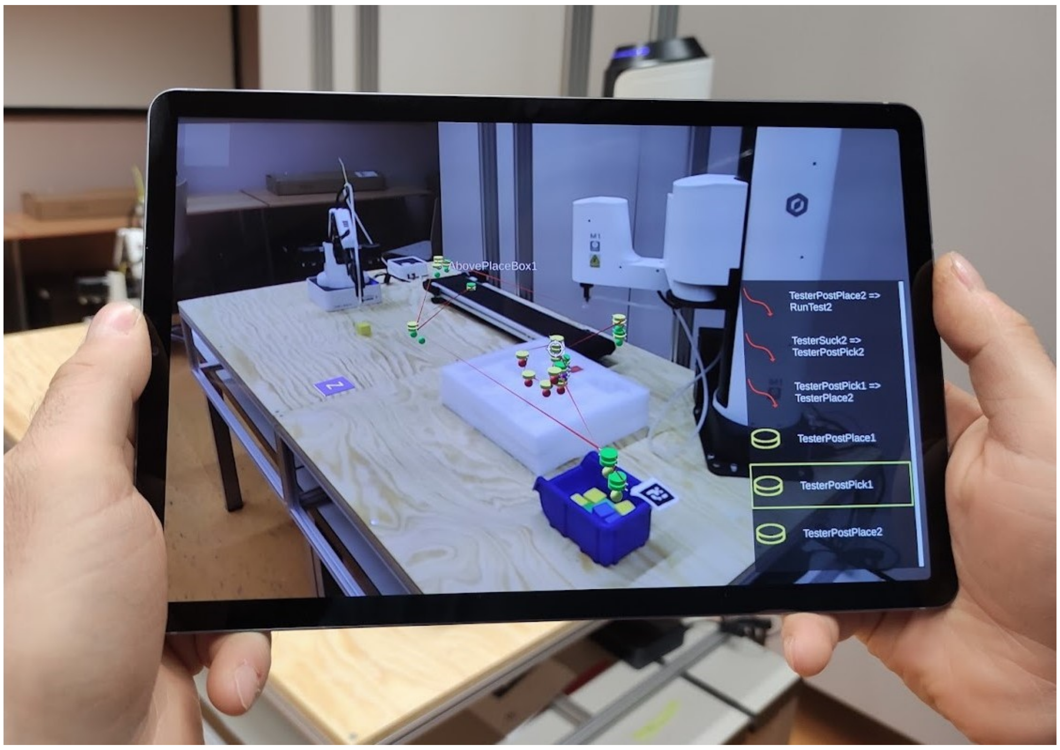

AbovePlaceBox1
TesterPostPlace2 => RunTest2
TesterSuck2 => TesterPostPick2
TesterPostPick1 => TesterPlace2
TesterPostPlace1
TesterPostPick1
TesterPostPlace2

Figure 9.4: Selector menu. The user selects the object by combining the device's physical movement and the object's selection using a side menu.

Spatial hierarchical menu

The first proposed method deals with the clustering of objects not only by distance but also by the other parameters of the object. In our case, the primary parameter was the class of the objects, which could be the scene object, spatial anchor, action, or connection of actions. Objects of all classes are presented in Fig. 7.10. The scene object is, for example, the blue box in the right bottom corner (which is, in fact, a real object but represented by the virtual box of the same size and position in the virtual scene). The spheres of various colors represent the action points, i.e., the important 3D space anchors, where some action should take place. The actions (program steps with the purpose indicated by different colors) are represented by the cylinders located above the spatial anchors, and the red lines represent the connections between actions.

Upon selecting a spatial cluster, the spatial hierarchical menu (see Fig. 9.3) appears in front of the user. The objects are divided into four categories, each represented by one quadrant of a full circle. The object's class in the quadrant is represented by a 2D drawing of the object of the selected class. If there are more than four classes, one of the quadrants is used as a container for other classes, through which the user could reach the remaining ones. Depending on the number of different objects' parameters in the scene, there could be more than four sections. Nevertheless, more sections increase the difficulty of the section selection and decrease the amount of space for rendering virtual representations of the objects in the section. Four sections (i.e., quadrants) were selected for our experimental task.

The hierarchical menu is recursive, meaning that after selecting one of the quadrants, all objects present in the selected quadrant are further divided into new quadrants based on selected parameters. These parameters are, again, application-dependent. For our evaluation, the color was selected as the second parameter. When there is only one object in the selected quadrant, it is returned to the application as selected. We have based this approach on SQUAD [74] technique, used for selection by progressive refinement of a cluster of objects based on spatial, visual, and other parameters.

A text label is rendered in the proximity of each quadrant, showing either the primary or secondary parameter of objects in the quadrant (i.e., class or color in our case) in the case of multiple objects in the same quadrant or the name of the object when there is only one.

The hierarchical menu is rendered in the 3D space at a certain distance in front of the device. The user selects the desired quadrant by the same crosshair used to select the initial cluster and confirms the selection by clicking on the same circular button in the lower right corner of the screen which has been used for the cluster selection. This limits the user's attention switches between the scene and a head-up display.

Selector menu

The second proposed selection method is similar to the icon-based selection presented by [121]. It shows the obtained cluster in the form of a list on the side of the screen (head-up like, see Fig. 9.4), easily reachable by the user's right thumb when holding the tablet device using both hands. Each item in the menu contains an icon representing the object's class, color, and name. The icons are the same as in the hierarchical menu above. The directly hit object is highlighted using the yellow outline in both the scene and the menu. The user makes the selection by touching one of the items in the list, regardless of whether the item is highlighted or not.

The list of the objects is continuously updated as the user hovers the tablet over the environment. A simple hysteresis supported the stability of the objects in the list, and updates were limited to 2 Hz to lower the flickering of the objects in the list.

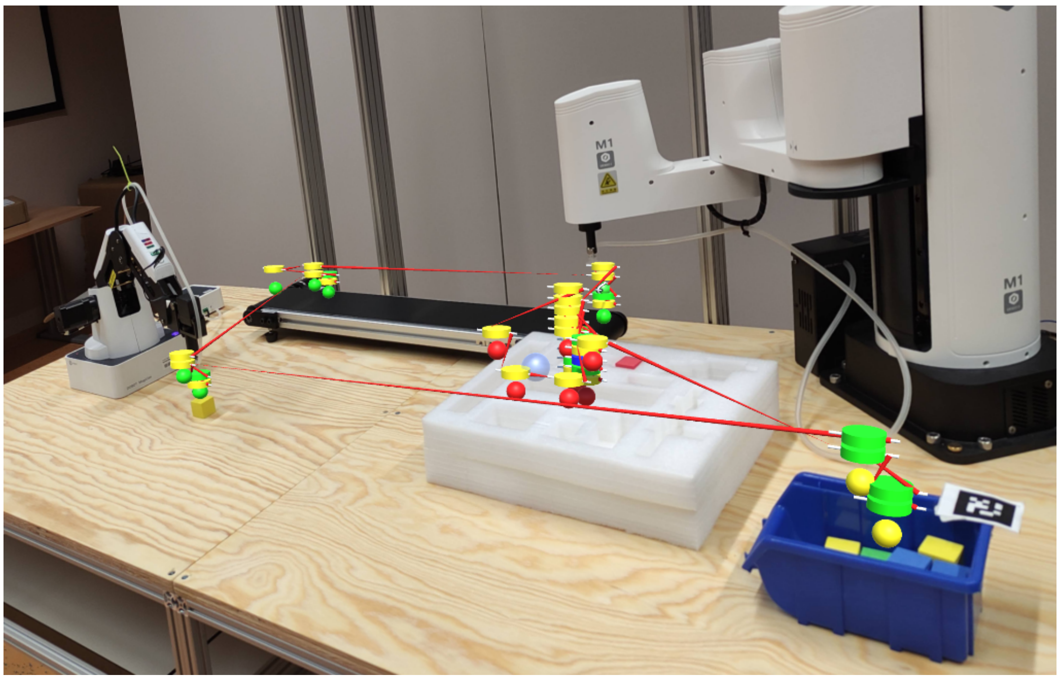


Figure 9.5: The virtual scene rendered over the real environment, used in conducted experiment. The virtual objects represent a simple program, where the small robot should pick a cube, move it to the conveyor belt and pass it to the second robot. The second robot should pick up the cube again, touch the simulated device (represented by a white foam box) in several places with the cube attached, and drop the cube into the blue box.

9.3 PILOT EXPERIMENT

The experiment was organized as a within-subject study with three methods in randomized order: direct touch – baseline (A), spatial hierarchical menu (B), and selector menu (C). It was carried out in a laboratory, on a demonstration workspace for a visual programming framework, equipped with two robots (Dobot Magician and Dobot M1) and a conveyor belt (see Fig. 9.5). Robots were switched on to support the robot programming feeling but remained stationary during the experiment.

The objects to be selected represent a simple pick and place program. A total of 91 virtual objects were displayed above the workplace, spread across the area of 1.3m^2, grouped into five clusters (see Fig. 9.5). For each tested selection method, the task was to select 30 objects (three scene objects, six action points, 21 actions) – those were the same for all methods, however, in a different order (same across the participants). The task inevitably contained a search phase (which Fitt's Law tries to avoid), as objects were spread across a large area and could not fit into the field of view. The users were allowed to move freely. However, the search phase was present for all methods and should not affect the results. The white outline

	age	AR_x	task time [s]			success rate [–]		
			A	B	C	A	B	C
p1	35	4	114.56	258.42	187.39	0.77	1.00	0.93
p2	25	2	135.93	380.55	240.26	0.73	0.90	0.93
p3	30	2	142.46	319.42	311.92	0.83	0.90	0.80
mean	30	2.67	*130.98*	319.46	246.52	0.78	*0.93*	0.89

Table 9.1: Demographic information about participants (AR_x denotes experience with AR on the scale of 1-5.) and obtained data, objective measures task time (total time to select 30 objects) and success rate (in the range of $[0, 1]$).

highlighted the object to be selected, and the object's name was rendered next to it. Therefore it was visible even from a distance.

We chose *task time*, *success rate* (both representing task performance) and *trajectory* (distance reported by the tablet's visual tracking, therefore having limited precision; could be related to necessary physical effort) as objective metrics and NASA-TLX [54] as a subjective metric.

Participants were recruited from faculty staff (3 males, one Ph.D. student, and two postdocs), further denoted as p_{1-3}. They were first informed about the study purpose and signed informed consent. The moderator explained the usage of each method, and participants were asked to select four objects as training for each method. After the training session, they performed 30 selections and filled in the NASA-TLX questionnaire for the method. In the end, the moderator carried out a debriefing with the participant.

The order of methods assigned to participants was $p_1 := \{A, B, C\}$, $p_2 := \{B, C, A\}$ and $p_3 := \{C, A, B\}$.

9.4 RESULTS AND DISCUSSION

The results indicate that both indirect methods provide better success rates than the baseline, however, at the expense of notably longer times. Those results seem to be consistent between participants. The big difference in the trajectory metric for all methods for the participant p_3 was probably caused by his adopted interaction strategy. Most of the time, the first two participants stood in one place, while the last one walked around the workplace to acquire the best pose for selection. All measured data are presented in Table 9.1 and Table 9.2.

Regarding the NASA-TLX metric, all participants ranked the A method as the one with the lowest task load (see Table 9.2), which could be influenced by the general

	trajectory [m]			TLX [–]		
	A	B	C	A	B	C
p1	15.40	26.60	20.10	19.44	66.67	41.67
p2	16.44	24.11	15.80	2.78	11.11	13.89
p3	39.78	40.96	32.66	2.78	25.00	16.67
mean	23.87	30.56	*22.85*	*8.33*	34.26	24.08

Table 9.2: Obtained data, where objective measure is trajectory (total device displacement as measured by visual tracking) and subjective measure is NASA-TLX (range $[0, 100]$, the lower the better).

acquaintance of the baseline method. For the following research, the training session will be extended and improved. We observed that most of the participants' problems occurred during the first few selection attempts of the main task, which could influence the results.

During a debriefing, the p3 stated that he felt confident when using the C and especially the B method, as he was informed of which objects were about to be selected. In the A method, he was sometimes unsure because of the small size and occlusion of the target. Participants p1 and p2 both complained about the instability of the objects in the list. The participant p2 suggested a freeze button for the left thumb, which will pause any changes in the selector menu and help him comfortably select the desired object. Alternatively, adopting some temporally stable labeling methods, such as the one presented by Bobak et al. [22], could significantly improve the stability of objects in the list. The p2 generally liked method A, but he complained about the need to move close to the object to achieve a certain degree of accuracy.

The participants p1 and p3 held the device with one hand on the short side while using the other hand for object selections. From our experience, such holding of a tablet device causes arm fatigue in the case of longer sessions. Also, we identified a few usability problems in the indirect methods' design, which might impact results. For B, participants p2 and p3 sometimes had problems stepping back in the hierarchy menu, and they accidentally selected the wrong object instead, causing a worse success ratio. Moreover, the labels of the quadrants were too small and thus hard to read. In the case of C, the instability of the order of menu items probably caused some wrong selections and could lead to a higher task load. According to observations, users tend to precisely aim at an object with a crosshair to get it highlighted in the side menu, although they could select any object listed there. We speculate that removing the highlighting mechanism from the menu and instead highlighting the whole cluster (the content of the menu) in the scene will make the method faster and improve its usability.

9.5 CONCLUSIONS

The conducted experiment provides the initial comparison of three different methods for virtual object selection, one direct and two indirect, to further enhance to interaction with the system, as required by O1. The purpose of this pilot experiment was mainly to obtain first impressions and validate the study design.

A preliminary evaluation of the new methods suggested that using indirect methods on mobile devices in AR could help increase selection accuracy. The increase could be achieved primarily in tasks with heavily cluttered environments, such as robot visual programming, AR-enabled visualization of robotic trajectories, editor of robotics work cells, or any other situated visualization. Valuable feedback for my prototype was collected, which will be addressed in a refined version of the proposed methods, and thoroughly evaluated later.

A significantly higher number of participants will be involved to observe statistical differences between methods for the final experiment. The experiment design will be slightly altered, as the training session will be extended, and the methods will be explained more thoroughly to the participants.

10 FRAMEWORK FOR COLLABORATIVE END-USER MANAGEMENT OF INDUSTRIAL ROBOTIC WORKPLACES USING AUGMENTED REALITY

Previous chapters described the concept of simplified robot programming using the AR on mobile devices and provided evaluation of the SAA concept using a prototype application. Following the previous research, we have developed a fully functional solution to evaluate the concept more thoroughly as the O4 requires. The ARCOR2[1] framework, including integration of selected robots from ABB, Fanuc, or Dobot, was created and published as open source on github[2]. This framework enables end-users to perform complete management of a robotic workplace or a production line, such as an initial setup, programming, adaptation, releasing to production, or controlling execution. Its user interface can be seen as a universal teach pendant for all robot types, machines, or APIs where a new device or service can be integrated by writing a custom plugin in Python. Even if the device does not profit from the spatial nature of the robotics program, the user will benefit from a unified interface. This integrative approach eliminates the need to undergo training for the interface of each device involved. The user interface is designed for commodity tablets and utilizes AR to visualize program data, including spatial points, program instructions, and a logic flow. One tablet can be used to manage multiple workplaces.

This chapter is an extended reprint of a paper [71], which is submitted for review in the Journal of Intelligent & Robotic Systems at the moment of writing this thesis. The paper presents the research and development of a fully functional prototype of a simplified robot programming tool, using the spatially anchored actions presented in the previous chapters.

10.1 RELATED WORK

Although some off-the-shelf teach pendants offer a particular form of simplified programming, the usability seems relatively low [125]. The missing visualization, inability to use common syntax structures as conditions and loops, high mental

[1] Augmented Reality COlaborative Robot

[2] https://github.com/robofit/arcor2 and https://github.com/robofit/arcor2_areditor

and physical demands, or lack of tools for debugging [6, 59, 64, 150] seem to be the main issues.

Moreover, pendants are often vendor-specific and limited to programming robots. However, there are typically more devices in the work cell that must be programmed separately and somehow synchronized with the robot. On the other hand, offline programming tools such as RoboDK or ABB RobotStudio offer comprehensive functionality but require extensive training. Additionally, PC-based and pendant-like user interfaces are, in general, likely not optimal for end-user programming, as they imply continuous switching of a user's attention [150] and therefore induce high cognitive and attention-related workload. Kinesthetic teaching is often employed to allow users to set target poses or waypoints intuitively and simplify programming. However, depending on the stiffness and size of a particular robot, it could be physically demanding and not desirable to users [6, 59].

Proposed Approaches

Many alternative methods for simplified programming and even third-party complete solutions were proposed. Some of them are not intended as comprehensive tools but rather focused on a specific task, aspect of the process, or are limited to a certain robot. The simplification is usually achieved through some form of visual programming [46, 62, 94, 104, 136], spatial visualization enabling the user to work within the task context [46, 143, 156], commonly combined with a kinesthetic teaching [80, 104, 115] and/or perception [62].

Paxton et al. [104], for example, proposed a "CoSTAR: the Collaborative System for Task Automation and Recognition" (see Fig. 10.1). They have employed a Behavior Tree-based task editor, combining high-level information from known object segmentation with spatial reasoning and robot actions. By incorporating the perception methods, the CoSTAR system can work robustly, as it can adapt to changes in the scene. Still, it could be operated by non-experts due to the usage of visual programming.

Huang et al. [62] proposed another system for simplified robot programming, called Code3 (see Fig. 10.2). It consists of a Blockly-based code editor based on visual programming, a perception module for object detection, and a programming-by-demonstration module for defining robot movements. The experimental evaluation has shown that the users can program useful programs after 90 minutes of training with the system.

Unfortunately, there still exist many limitations. Only a tiny portion of evaluations are carried out on nontrivial use-cases as in [94], or contain comparison with an existing method as in [46, 106, 136]. Often, there is only a simplified method

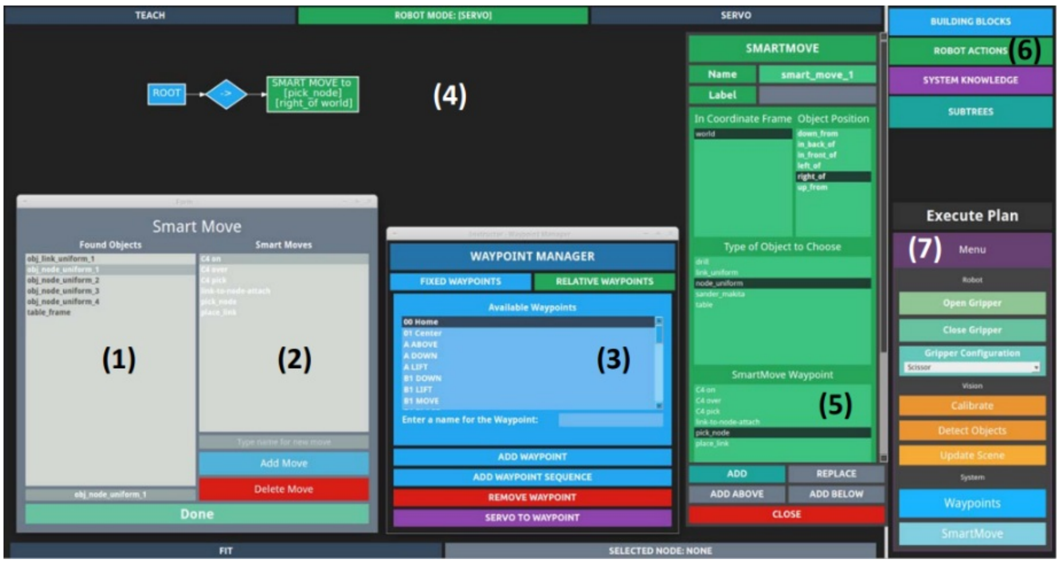
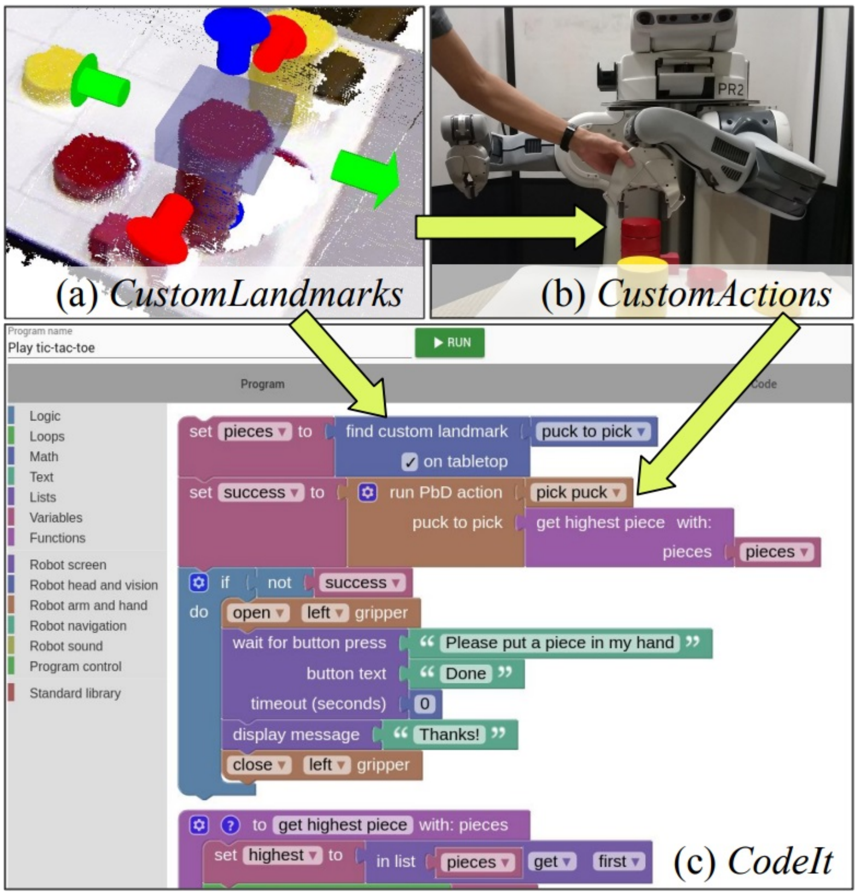


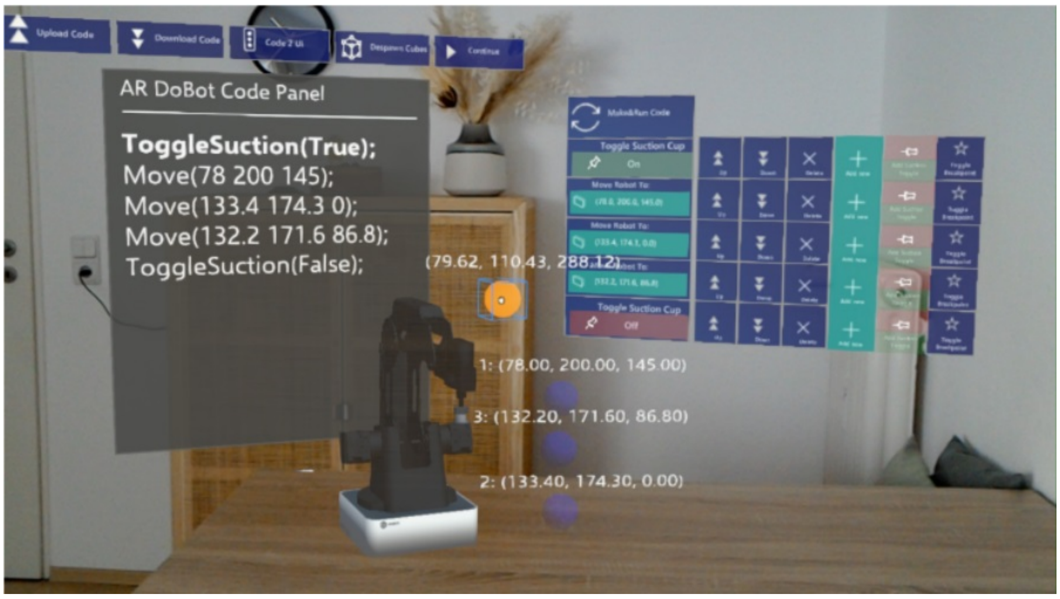
Figure 10.1: Behavior Tree-based user interface. (1,2) Detected objects and associated SmartMoves. (3) Waypoint UI. (4) User's workspace containing the Behavior Tree. (5) SmartMove creation pane; Similar panes allow customization of other operations. (6) List of available operations. (7) Expanded menu containing simple operations the user can perform during plan development. Reprint from [104].



```
set pieces to find custom landmark puck to pick
                 on tabletop
set success to run PbD action pick puck
                 puck to pick get highest piece with:
                                 pieces pieces
if not success
do open left gripper
   wait for button press "Please put a piece in my hand"
              button text "Done"
        timeout (seconds) 0
   display message "Thanks!"
   close left gripper

to get highest piece with: pieces
   set highest to in list pieces get first
```

Figure 10.2: Overview of the Code3 system. Reprint from [62].



```
ToggleSuction(True);
Move(78 200 145);
Move(133.4 174.3 0);
Move(132.2 171.6 86.8);
ToggleSuction(False);
```

Figure 10.3: Robot Code, Robot Model in an AR application and block-based DSL Code (f.l.t.r.). Reprint from [62].

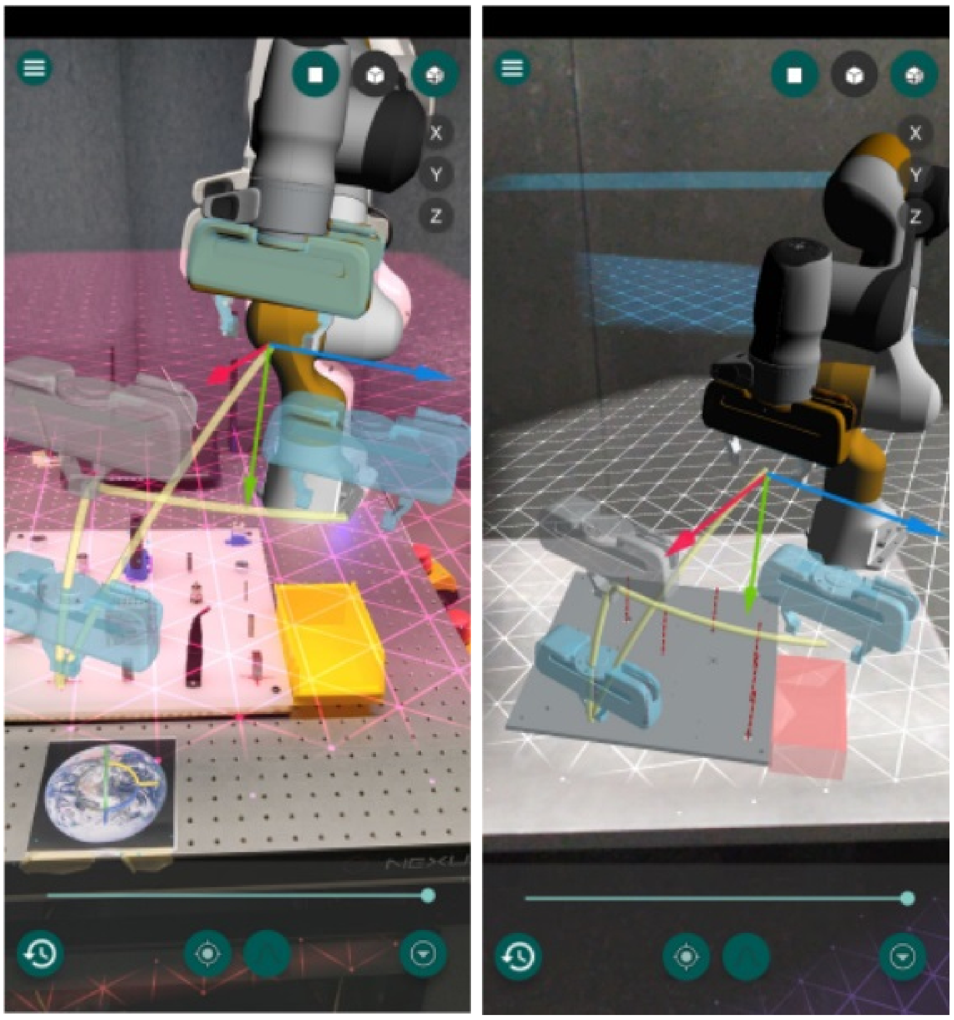
available, which precludes the possibility of (remote) expert intervention, where it can be assumed that the expert prefers to work with source code. The issue could be, for instance, solved by generating the source code from visual representation [76, 156] and probably optimally by bidirectional synchronization between those two.

Visualization Methods

AR seems to be a promising visualization method for simplified programming on a high level of abstraction. It may enable users to work within the task space and avoid superfluous attention switches, mental transformations, and related workload. However, only a few approaches allow programming in AR [46, 156, 158] or provide the ability to set up a workspace [143] and therefore do not require an additional intermediate user interface.

The SPEARED interface (see Fig. 10.3), proposed by Yigitbas et al. [156], provides an AR application on Microsoft HoloLens HMD for robot programming. It shows the code as draggable blocks, while the spatial parameters are displayed as textual coordinates and spheres in the environment. According to their evaluation, this kind of interface has the potential for non-experts to program robots or other smart devices.

Thoo et al. [143] have proposed another AR method of robot programming using smartphones or tablets (see Fig. 10.4), enabling the user to model a virtual representation of a workspace, which enables the user to create or adapt robotic tasks, without the need to shut down the actual robot's workspace for the pro-



(a) Illustration of the ability to program the robot offline directly in the real robot's workspace.

(b) Illustration of the ability to program the robot offline in a virtual workspace by defining key points (displayed as transparent end-effectors).

Figure 10.4: The user interface of the AR robot programming system proposed by Thoo et al. [143], showing the programming interface and visualizing the resulting trajectory. Reprint from [143].

gramming purposes. As shown in Fig. 10.4, the user could work directly in the actual workspace or a virtual workspace. It enables the user to control the robot in real-time to program it offline.

Often, AR is used only as an extension, e.g., to visualize robot trajectories. In general, there is a lack of tools for precise manipulation of robot or virtual elements, which is necessary for industrial use cases. The AR may be, for instance, realized by spatial projection [46, 91], which is limited to visualization on surfaces. Head-up stereoscopic displays can convey depth but, on the other hand, are expensive, offer a limited field of view, and require learning of unconventional control (e.g., gestures). Usage of hand-held devices leads to problems with missing depth perception [143]; however, those are affordable, portable, and easy to use.

Summary

As seen from the previous related work overview, many approaches exist to lower the barrier to entry to robot programming by various means. However, there is a lack of comprehensive yet simple environments, allowing end-users to perform all tasks and steps necessary for industrial-like applications. Also, it has to be considered that modern workplaces may contain not only a robot but multiple (programmable) machines or special-purpose devices. With Industry 4.0, there will also be a rising need to communicate with various services through their APIs.

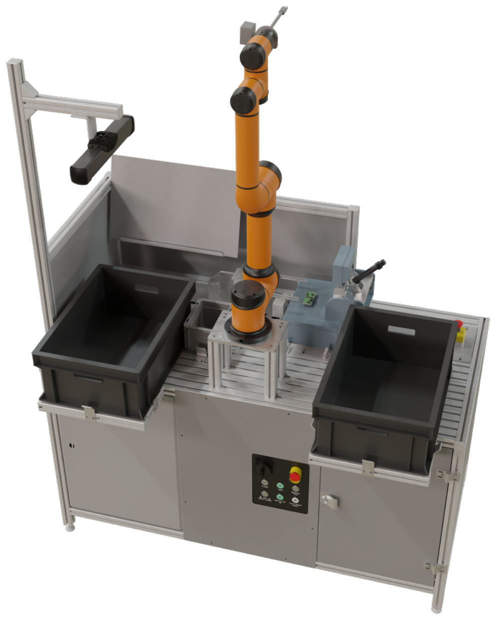


Figure 10.5: Render of a PCB testing workplace with the Ensenso 3D camera for bin-picking, 6 DoF Aubo i5 robot, 2 DoF custom-build robot, functional tester, barcode reader, and printer, source, and target boxes.

10.2 USE CASE

Although the framework was designed to be general, it was made with our scenario 2 (presented in Section 3.2) in mind. In the case of the traditional approach, the 6 DoF robot would be programmed using a teach pendant, and a PLC would operate all the other devices. A highly trained operator would be needed to adapt such a heterogeneous workplace to a new product. In the proposed approach, the system integrator will develop an integration for all devices into the ARCOR2 system, providing functionality on the optimal level of abstraction for the task. The system integrator will also do the initial setup of each workplace (for its visualization, see Fig. 10.5). Then, changes can be either done by a trained operator or remotely by an expert programmer. The main advantage for the end-user is that there is just one configuration, programming, visualization, and control interface.

Based on a comprehensive discussion with the project partner, a set of requirements for the system were defined:

1. Convenient integration of new robots, machines, and services with variable levels of abstraction.
2. Support collaboration between end-users and experts.
3. Ability to manage (perform CRUD[3] operations on):
   a) Setups of the workplaces (available objects, their locations, and parameters).
   b) Important points in space and associated data.
   c) Program steps and their parameters.
   d) Self-contained executable snapshots of programs.
4. Robot as a source of precision.
5. Control and visualization of an execution state.
6. Debugging capabilities.

A set of different user roles were also defined. They were divided into two categories and can be seen on Section 10.2. For each role, there are different responsibilities and needs.

[3] Create, Read, Update, Delete

Category	Role	Responsibilities	Principal needs
End User	Operator	Manages program execution, solves simple problems.	Visualization of execution state, controls to start/stop program, notifications on errors.
	Standard User	Able to create a simple program or adapt an existing one.	Program management (edit, copy, etc.), tools to edit spatial points and program steps.
	Advanced User	Able to create complex program visually, can write simple code.	Visual definition of advanced concepts, simple and well-documented programming API.
Expert	Technician	Performs initial setup, called when serious problem occurs.	Debugging tools, entering exact numbers.
	Programmer	Integrates new devices, creates new functionality.	Well-defomed processes, generality and reusability of code

Table 10.1: Expected types of users, divided into two main categories.

10.3 SYSTEM DESIGN

The defined requirements mainly give the design of the framework. However, it was also influenced by the knowledge gained during the development and evaluations of its previous generation called ARCOR [91] (presented in the Chapter 5). It utilized SAR and a touch-enabled table for user interaction and was focused mainly on table-top scenarios. Although ARCOR was successfully evaluated in an industrial use case [68], its limitations (mainly complicated integration of new devices and program instructions) lead us to the development of the next generation, based on the Spatially Anchored Actions. It uses tablet devices, which allows easier integration into existing facilities, as there is no need for projectors or touch-enabled surfaces.

Terminology

The following list defines a set of terms used in the following chapters and their descriptions.

Object Type: Plugin into the system that represents and provides integration with a particular type of real-world object, e.g., a specific type of robot or a virtual object such as cloud API. It is written in Python and can benefit from (multiple) inheritances in order to extend or share functionality. A set of built-in base classes is available, representing, e.g., a generic robot or a camera and its required API. It could be associated with a model (representing both collision and visual geometry), which might be a geometric primitive or a mesh.

Action Object: An instance of an *Object Type* within the workplace, defined by its unique ID (UID), human-readable name, pose (optionally), and parameters (e.g. API URI, serial port, etc.).

Scene: A set of *Action Objects* with defined poses, represents a workplace, its objects and spatial relations.

Action Point: A spatially anchored container for orientations, robot joints configurations, and actions. The container's position and orientation comprise a usable pose, e.g., as a parameter for robot action.

Action: Method of an *Object Type* exposed to the AR environment. A named and parameterized action is called an action instance. Actions may be implemented on different levels of complexity according to the application needs and the target end-users competencies. However, to lower program complexity and reduce training time, the actions should be preferably high-level and provide configurable skill-like functionality. Such actions can be seen as equivalent to reusable skills used, e.g., in [136, 80].

Project: Set of *Action Points*, *Actions* and logic definition. The project is always associated with a *Scene*, as the *Action Objects* in the *Scene* defines the possible set of *Actions* used in the *Project*.

Execution Package: A self-contained executable snapshot of a *Project*, which is created when there is a need to test the whole task or release a project into a production environment. The fact that the *Package* is self-contained allows users to make further changes in the *Scene* or *Project* without any influence on already existing *Packages*.

Main Script: Contains a logic of the *Project*, which may be defined visually or could be written manually with the help of a set of generated classes providing access to *Project* data as, e.g., defined *Action Points*.

10.4 INTEGRATING NEW DEVICES

A new device is integrated into the system by implementing an *Object Type* (Python wrapper) for it that is based on some of the provided abstract base classes and is dynamically loaded into the system.

For instance, there is an abstract *Robot* class, and all *Object Types* representing particular robots are derived from it. It has a set of basic abstract methods representing mandatory, or robot's minimal functionality (e.g., a method to get the end effector pose), that must be implemented. Then, there is a set of methods that may or may not be implemented based on the available functionality of the particular robot (e.g., a method for forward and inverse kinematics or for toggling the hand teaching mode). After the wrapper is loaded, the system performs static analysis to determine in advance which optional methods are available. Based on that, certain functionality is or is not made available to the user.

There are two main possibilities of how an *Object Type* could be interfaced with a real-world object, e.g., a robot:

1. Directly — if the robot provides API with the necessary functionality, the *Object Type* (i.e., the python plugin) may communicate with it directly.
2. Through an intermediate service – for instance, if the robot lacks motion planning capability, there might be a ROS-based container between the robot and the *Object Type*.

In both cases, the *Object Type* is the main provider of all *Actions*, available to the user, regardless of the interface between the *Object Type* and the real world object.

10.5 ARCHITECTURE

The framework is divided into services (backend) and a user interface (frontend). The main service of the system is **ARServer**, which acts as a central point for user interfaces and mediates communication with other services (see Fig. 10.6).

So far, two implementations of a user interface have been developed:

1. A tablet-based app provides the full functionality of the framework.
2. A Microsoft HoloLens app provides a basic set of operations, such as manipulation with *Action Objects* or *Action Points* and visualization of the whole program in an immersive way using a HMD.

The intention is to allow the involvement of several simpler, complementary interfaces providing only some aspect of functionality. One example is an RGB LED strip indicating system status; another is a hand tracking-based interface for controlling a robot. A projected interface similar to the one presented in previous chapters could also be incorporated, providing crucial system information without the need to wear a HMD or hold a tablet. Therefore, the server must be able to deal with multiple connected interfaces, even in single-user scenarios.

Interfaces are connected to *ARServer* using Websockets, which allows bidirectional communication. The *ARServer* holds the system state, while interfaces can manipulate it using a set of RPCs and receive notifications on changes. It is assumed that each workplace runs its instance of *ARServer* and therefore, the server maintains only one session for all users. If one user opens a Project, the same

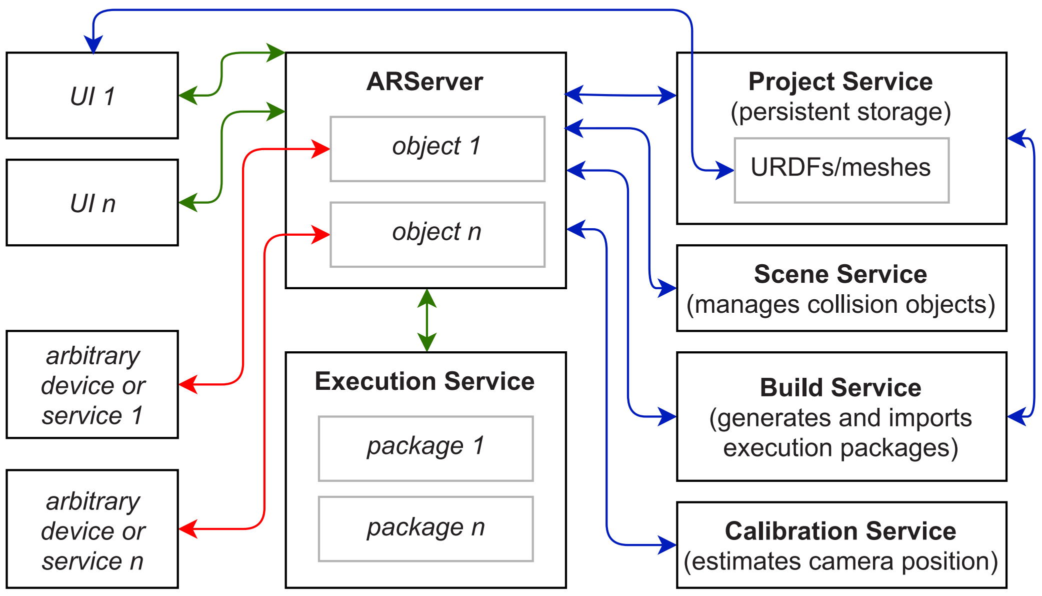


Figure 10.6: Block diagram of the system in a state when object instances are created in the ARServer (scene/project opened and online). Green lines depict WebSockets connections (two-way communication necessary), blue are REST APIs, and for red, an implementer is free to choose appropriate technology.

Project is shown to other connected users. In order to support efficient and safe collaboration between users, there is a locking mechanism that prevents multiple users from manipulating the same element (e.g., controlling the robot).

The *ARServer* also serves as a proxy between Python code and AR environment, which is code-agnostic. It analyzes the code of Object Types in order to extract available Actions and their parameters and creates JSON metadata that is available to user interfaces. The code analysis takes advantage of PEP 484 type hints[4] in order to extract parameter types and matching nodes of Abstract Syntax Tree (AST)[5], among others. It allows to, e.g., inspect value ranges that are defined using assertions or check if a method (feature) is implemented.

The *Scene* or *Project* opened within the server could either be in an offline or online state. In the online state, instances for all objects are created on the *ARServer*, meaning that, e.g., a connection to a robot is successfully made. In an online state, robots could be manipulated and any action instance added to a project may be executed, simplifying the programming and debugging process. However, it is also possible to work offline, where certain functionalities such as controlling a robot are unavailable. Moreover, in the offline mode, the robot and other relevant machines do not have to be connected. Therefore, the operator may prepare the base program in advance without needing the actual robot.

[4] https://peps.python.org/pep-0484/

[5] https://docs.python.org/3.8/library/ast.html

Project Service provides persistent storage for workplace-relevant data, such as *Scenes*, *Projects*, *Object Types*, *3D objects models* (STL, Collada and others), etc. It is accessible using a REST interface.

Scene Service, used, e.g., in cases where underlying implementation is ROS-based, is responsible for the management of collision objects.

The **Build Service** creates for a given *Project* a self-contained *Executable package*. The logic could be defined within the AR environment or provided in a standalone python script using a project-defined *Action Point* to define spatial information. When generating logic from its JSON representation, it is first assembled in the form of AST and then compiled into Python code. Moreover, a set of supplementary classes, e.g., simplifying work with *Action Points* are generated.

Execution Service manages execution packages created by the *Build Service*. The most important functionality is running the *Package* when the service streams events regarding execution state (e.g. which *Action* with what parameters is being executed) to *ARServer*. The execution can also be paused or resumed when needed. Basic debugging functionality, such as stepping over individual *Actions* or breakpoints defined on *Action Points* (meaning the program will be paused when an *Action* which uses a selected *Action Point* as a parameter is being executed) is supported.

Calibration Service provides a method to perform camera pose estimation based on ArUco marker detection [118]. The service is configured with IDs, poses, and sizes of available markers. When estimation is requested, markers are detected in the provided image, respective camera poses are computed, and then all poses are averaged using a camera-marker distance and camera-marker orientation as weights to produce the final 6D pose. A method from [89] is used for averaging quaternions. This estimation can be either used by user interfaces where, e.g., an AR visualization needs to be globally anchored, or it could be used by *ARServer* to update the pose of the camera in a scene. Another service method may be used to adjust the robot's pose using an RGBD camera. The virtual robot model in a configuration corresponding to the actual robot state is rendered from the point of view of a camera in a robot's current position within the scene, which therefore serves as an initial guess. The virtual camera is used to generate a point cloud registered using a robust ICP (TukeyLoss kernel) with the point cloud from the real RGBD camera observing the scene (1024 frames averaged) that has been filtered to contain only close surroundings around the robot in its initial pose. If the precision of such calibration is not enough for the task, more precise methods must be used, and the robot's or camera's pose can be entered manually.

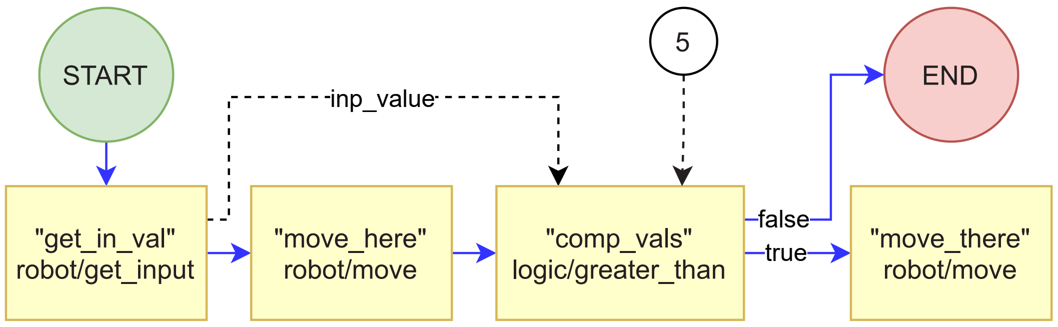


Figure 10.7: The logical structure of an example program. Yellow boxes are Actions (text in quotes is the user-entered name for the instance of Action, below is an object and the corresponding method), blue lines denote logical connections (flow of the program), and black dashed lines denote data connections. The example shows how previous results can be used as parameters of a subsequent action and how logical flow can be branched based on a numerable value (boolean in this case).

Program representation

One of the framework's goals is to support collaboration between non-programmers who prefer creating programs visually and programmers who prefer to work with code. Because of this, there are two language representations. There is an intermediate program representation for visual programming based on JSON format, which is language agnostic and easy to serialize. Moreover, it supports standard programming techniques (cycles, conditions, variables), allows flexible parameter specification, and is easily manipulable from user interfaces. For execution, the intermediate format is translated into Python, which is currently the most popular scripting language[6]. The same language is also used to implement *Object Types*, through which a new device can be integrated into the system. It also allows a use case where a non-expert user creates the program visually, and an expert programmer adjusts the resulted Python script. The form of Python code was designed with the possibility of transferring the code back into the intermediate format. However, this was not implemented yet.

The structure of the JSON format is as follows. Within a *Project*, there might be $[0, n]$ *Action Points*, where each might contain $[0, n]$ actions. Each action is assigned a UID, unique human-readable name, type (*Scene object* UID and corresponding underlying method/*Action*), and $[0, n]$ parameters (corresponding to parameters of the method). *Action* parameters can be given as literal or referenced to either a project variable (constant shared by multiple actions) or a previous result (return value of precedent action). On the *Project* level, there is an array of objects

[6] According to PYPL Index for July 2022.

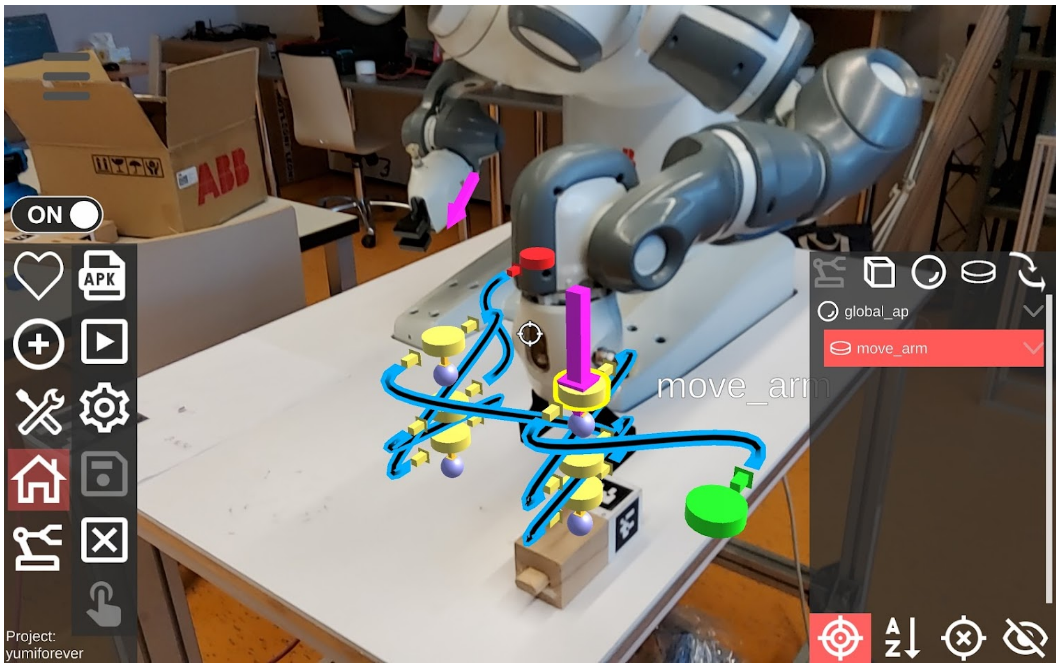


Figure 10.8: The application screen with a tool menu (left), selector menu (right) and non-interactive 3D scene (center), with action points (violet), actions (yellow, green for program loop start, red for its end), logical connections (black/blue) and robot end effectors (magenta). The visualized program realizes a simple pick and place task consisting of low-level actions.

defining logical structure (visualized as blue lines, see Fig. 10.8), where each contains UID of source and target *Action* and optional condition. *Actions* together with those linkages form a directed acyclic graph, where the loops are forbidden at an application level. Without a condition, two *Actions* could be connected only with one logic linkage. Conditions are meant to achieve simple branching for numerable types such as boolean, enums, and integers. E.g., for branching according to a boolean value, two logical linkages are added, one for true and the other for false (see Fig. 10.7 and Listing 10.1). Any other type of condition has to be implemented in a form of *Action*, for instance, *greater than(float1, float2)* returning a boolean value. Also, loops are not part of the format definition and have to be implemented in the form of custom actions. This restriction keeps the intermediate format simple and simultaneously allows integrators to provide a customized set of *Actions* to their end-users.

```python
inp_value = robot.get_input(an="get_in_val")
robot.move(an="move_here")
comp_res = logic.greater_than(inp_value, 5, an="comp")

if comp_res == True:
  robot.move(an="move_there")
```

Listing 10.1: An example of generated Python code. Parameter an denotes action name, which is human-readable counterpart to action UID.

User Interaction

Based on the concept presented in the previous chapter, a working prototype was implemented, iteratively tested, and improved in cooperation with the project partner. The application's design was modified to support the two-handed operation of the tablet and control of interface elements using the user's thumbs, to lower the fatigue of arms and hands.

The primary concept of the tablet user interface deals with the fact that most robotic programs interact with a real environment in some manner. Using the user interface, an operator can annotate the environment in a simplified way and subsequently design programs' logic. Thanks to the utilization of AR, this can happen within the task space, and therefore mental demands are lowered [59, 158].

The user interface uses several graphical elements for precise annotation of specific places in the environment (*Action points*). These places may later be used as spatial anchors for elements representing specific *Actions*. Visual elements representing *Actions* (formerly known as *pucks* in our GUI) are therefore located at the place where the *Action* will be executed, which improves users' comprehension of spatial relations within the program.

The interface consists of three main parts: the sight in the middle of the screen, the main menu on the left, and the tool context menu on the right. The sight is used to select virtual objects by the physical pointing of the tablet in combination with the selector menu, presented and initially evaluated in the previous chapter. The tool selection menu shows actions for a currently selected object (e.g., duplicate object or transform object). The tool context menu serves as a sub-menu for currently performed operations (e.g., move / rotate tools when the action object is being manipulated). See Fig. 10.8.

10.6 EVALUATION

During development, the system and its user interface were evaluated multiple times using different methods and continuously refined to provide a plausible user experience and fulfill use case needs. The evaluations of the interface concept and the selection method were presented in previous chapters. Other evaluations are described here.

Non-immersive VR mode

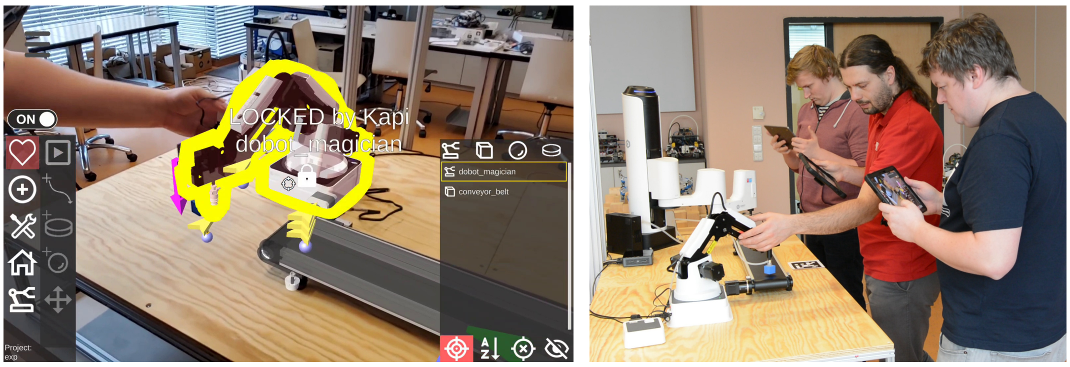
When working with AR, there is often a need to move closer in order to distinguish or inspect virtual objects or, in contrast, to move further in order to see the whole scene, which is amplified by the limited field of view of handheld devices and HMD. Regarding handheld devices, it is also often necessary to see the scene from different angles, to correctly judge the placement of the objects (or the distance from the user), which is caused by a lack of depth perception due to monoscopic display. Moreover, industrial environments typically have limited floor space because of, e.g., safety curtains or fences around robot cells. These constraints might make viewing the workplace from certain poses physically challenging or even impossible. Therefore we proposed and evaluated an approach allowing temporal switching from AR to a non-immersive VR [12]. In VR, the application shows a 3D model of the workplace, and the viewpoint is controlled by device motion in conjunction with on-screen joysticks, with non-linear sensitivity. The conducted experiment ($n = 20$), based on the object alignment task, revealed that self-reported physical demands are significantly lower when users are allowed to arbitrarily switch between AR and VR. The usefulness of the VR mode was rated as high during the task, and users spent 70% of the time within it. Observations of users' behavior have revealed that the VR mode was often used to get an overview of the workspace, to find an occluded object, or to avoid an uncomfortable position.

Multi-user Collaboration

In order to evaluate the collaborative aspects of the framework, a small-scale user experiment ($n = 3$) was carried out in a lab-like environment (see Fig. 10.9). The experiment was focused on functionality; however, it also served as the very first usability evaluation. The workplace consisted of two robots (Dobot M1 and Magician), a conveyor belt, and several collision objects.

The task was to set up the workplace collaboratively and to create a simple program for moving cubes from one robot to the other and back using the conveyor belt. In the setup phase, each participant added one *Action Object* (a robot or the belt) to the *Scene* and positioned it. Subsequently, a *Project* was created and each participant created several *Action Points* (using a hand teaching and visual positioning tools) and related *Actions*. Finally, the logical connections defining the program flow were added, and the Project was executed. The participants worked on separate sub-tasks most of the time but shared the same workspace. Moreover, they had to collaborate to connect all sub-tasks into a working program successfully.

From the technical perspective, the user experience during the collaboration was smooth, and user interfaces were kept appropriately synchronized. Regarding usability, although users communicated verbally during the experiment, they also appreciated visual indication of which object was being used (locked) by someone else. Based on the course of the experiment, collaborative programming seems to



(a) During hand teaching, the robot is locked by the user and therefore unavailable for others, which is indicated within the 3d scene.

(b) Multiple users collaborating on the task of moving boxes between robots using a conveyor belt.

Figure 10.9: Technically oriented evaluation of the collaborative-related functionality.

be a viable approach, and we will investigate it more deeply in future research. In this case, the task was done by three users in approximately 15 minutes. It would be interesting to determine the relation between task time and the number of collaborating users on a significantly more complex task.

Iterative Testing and Refinement

The whole system was created in close cooperation with our industrial partner, who has over 15 years of experience in automation. During the development, the design of the user interface and critical parts of the system were discussed in detail with the system integrator and potential end-users.

Besides individual testing, there were several integration meetings where the whole task programming process was tested with the actual production-like robotic cell to maximize simplicity and comprehensibility for the end-users. It helped us better understand real-world scenario difficulties, which are not evident during in-lab testing, e.g., object selection in heavily cluttered environments (e.g., caged robotic cell) or cooperation with safety precautions.

As a result, several improvements were added to the user interface. For example, a better ray-casting strategy for virtual object selection or enabling users to disable several objects (e.g., virtual safety walls), which have to be part of the Scene, but, once they are created, are barely used anymore.

Additionally, an unstructured interview with the system integrator representative was organized. They were asked to state their opinion on the framework being developed from the commercial perspective. Following key advantages were claimed:

- Clear visualization of position and distribution of individual actions in space.

- Ease and speed of programming for smaller-scale applications.
- Possibility of visual composition of the Scene - collision objects, positional relationships of individual elements.
- Simple and practical controls for the robot - stepping, end-effector alignment, integration of hand teaching.

Moreover, the following limitations were pointed out:

- Visual clutter for large-scale applications - too much graphical information.
- Ergonomically demanding method requiring the creation of the entire application in a standing position while holding a tablet.

Regarding the large-scale applications, it was suggested to define categories of actions to distinguish them, e.g., by color-coding or icons. Moreover, the visual clutter may be reduced by different techniques, such as implementing, e.g., level of detail [140], or by implementing more complex actions on a higher level of abstraction, which will reduce the number of individual actions needed to realize a given task. The proper holding of the device may improve the ergonomy of use (needs to be covered during training) [144] and also by already described VR mode. It allows users to temporarily switch from AR to VR to reach physically unreachable poses, zoom in, or work while sitting.

Expert review

When the system and interface design and features were mostly stable, an expert review was conducted to eliminate the most significant user experience problems and validate the system's overall concept. Three reviews were obtained.

The first reviewer (R_A) is an experienced software tester. The review was performed at the testing site of the project partner with an Aubo i5 and one custom-built two-axis robot. The second reviewer (R_B) is a 3D data visualization and processing specialist with experience in the field of HRI. The third one (R_C) is an expert in cyber-physical systems, computer graphics, user interface design, and evaluation.

Reviews by R_B and R_C were performed at the university robotics laboratory with an ABB YuMi robot. Reviewers R_A and R_B used the same version of the interface, while R_C used a slightly updated one that was available at the moment. Each session lasted approximately one and a half hours. The Samsung Galaxy Tab S6 with a protective cover was used. A reviewer was given a technical document describing the solution in advance and then briefly introduced to the usage of the

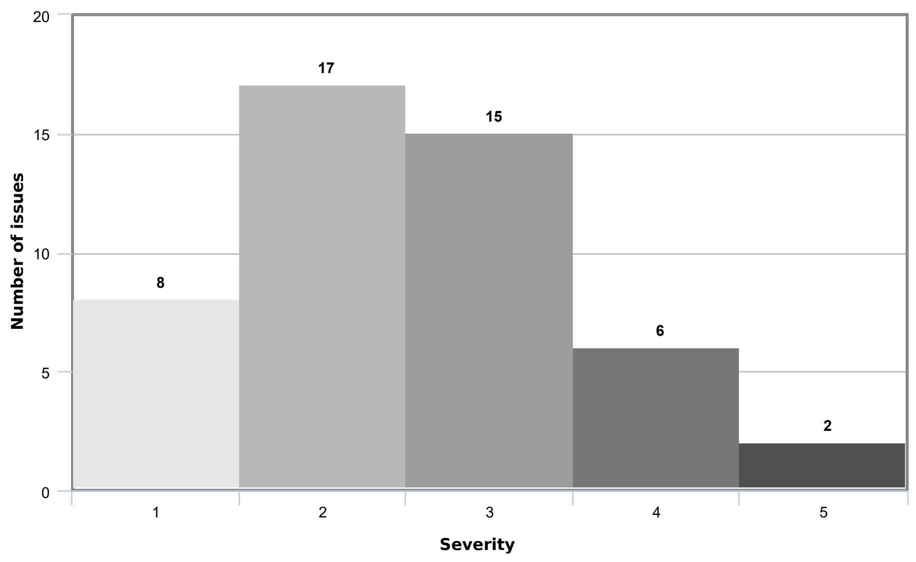


Figure 10.10: Histogram of all reported issues clustered by their severity.

interface. Then, they went through the core functionality while commenting on their findings. The comments were recorded and, after the experiment, processed into a review protocol. The reviewer was then asked to verify the protocol, briefly comment on each issue, and assign severity on a scale of $[1, 5]$. Thirty-nine unique issues (48 in total) of different severity (see Fig. 10.10) were identified by all reviewers. Most were ranked with low severity, dealing primarily with minor user interface usability issues. They include difficult number input (R_A) caused by the default Android keyboard, the unclear icon for a favorite group of actions (R_B), or issues with main menu actions grouping, forcing the reviewer to navigate through the menu to locate necessary action (R_C). The most severe issues are shown in Table 10.2, together with the reviewer's self-reported severity and brief suggestions.

The collected feedback was categorized into the following groups:

- **Control** (12 unique issues, 14 total) — issues related to user interface control.
- **HUD design** (13 unique issues, 17 in total) – problems with the user interface itself - icons, menu design, etc.
- **System Status** (3 unique issues, 4 in total) — related to notifications and system state visualization.
- **Visualization** (11 unique issues, 13 in total) — visualization of 3D scene content (*Action Points*, *Actions*, etc.).

The R_A suggested that there are too many icon buttons, and their purpose is not always apparent at first sight. The only way to understand individual icons is to hold a finger over them until a tooltip is shown. This issue will be solved by providing proper application documentation and onboarding mode, which will guide the novice user through the application.

The reviewer also pointed out that it was pretty physically demanding to hold the tablet for a longer period, and it was necessary to take rest periods regularly. It was partially caused by improper holding of the device because the R_A held it by one hand on the left side of the screen for some time at the beginning of the review, which caused fatigue to the arm. This issue could be addressed in the onboarding mode of the application, where proper holding should be demonstrated, and rest sessions suggested.

The testing site of the project partner contained virtual walls around the robotic cell assembled from dummy action objects. This complicated object selection because these safety walls were the first objects to be hit by the casted ray, making virtual objects inside the cell virtually impossible to select by aiming. This issue was already solved by enabling users to put selected virtual objects on the blocklist, thus excluding them from the selection process.

On the other hand, the R_A liked the visualization of the logical flow of the program, which helped them both understand the meaning of the edited program and change the program's behavior. Moreover, the reviewer appreciated the simplicity of robot stepping available directly from the programming application, without the necessity of using the dedicated teach pendant. The reviewer explicitly mentioned that the ability to align the robot's end-effector with the underlying table was crucial for the fast navigation of the robot.

The main issue for the second reviewer, the R_B, was related to depth sensing. The reviewer had problems with understanding where the manipulated object (e.g., *Action Point*) is in the real world, and they had to walk around heavily to see its position from multiple angles. They pointed out that it was probably caused by the lack of generated shadows, which usually helps people to sense the depth. As a (partial) solution, we have enabled shadows and light estimation in the application. To further support the user's knowledge of object position, a projection in the plane of the table, with information on the height above the table (or nearest object below), could be incorporated.

The reviewer also stated that the flow of the *Action* definition is unnecessarily complex and challenging, meaning that the user has to insert an *Action Point* first and then assign an *Action* to it. They suggested that simplification of this process would be a significant improvement.

The last but not least reviewer, the R_C, stated that it makes sense for them to define *Actions* inside the real environment. On the other hand, they were worried that it would be difficult to understand a more complex program, where a high amount of *Actions* together with conditional execution will be incorporated. The

	Issue	Severity	Suggestion	Category
R_A	Too much buttons and time-consuming determination of their meaning.	5	Implement a guide helping new users to get comfortable with application usage.	Documentation
	System status not visible in some cases (e.g., in case of long-lasting processes as calibration).	4	Add persistent notifications.	System Status
	Inability to aim objects through another object.	4	Add the possibility to disable an obstructing object temporarily.	Control
	Physical demands when working longer than 30 minutes.	4	Encourage both hand-holding, suggest rest period.	Control
R_B	A skill and much physical movement are needed to judge the position of virtual objects.	5	Add shadows to the virtual objects to improve depth perception.	Visualisation
	Complicated flow in order to add an action.	4	Allow to add an action without adding an action point first.	Control
R_C	Reachability of *action points* by selected robot is not visualised.	4	Add some indication of reachability.	Visualisation
	Difficult orientation in more complicated programs	4	Use different connections for different logic flow phases.	Visualisation

Table 10.2: The most severe issues (rated 4 or 5) and suggestions on how to mitigate them as reported by the reviewers.

reviewer suggested that selected logical flow parts should be differentiated by color or shape.

Another important issue for the R_C was that the robot's reachability of *Action Points* and executability of *Actions* was not visualized in any way. For *Actions*, a mark could be quite easily added to indicate whether or not the *Action* could be executed. In the case of the reachability of *Action Points*, the possibility of having more than one robot in the *Scene* needs to be considered. Moreover, any *Action Point* could potentially have several orientations (making several possible poses), and each of them needs to be evaluated separately.

In summary, the reviewers approved the overall concept of the framework as suitable for end-users. As they tested the framework through the user interface, naturally, most findings were related to its usability, where several relevant suggestions were collected.

10.7 CONCLUSIONS

The chapter presented the ARCOR2 framework, which allows end-users to program robots and complex workplaces or production lines consisting of multiple robots and other arbitrary machines. The framework was designed, developed, and iteratively tested in close cooperation with the industrial partner, who provided the real use case, testing site, and valuable feedback. One of the main advantages of the solution is that support for any device or service could be added in the form of a plugin. The visual programming in the AR interface allows specifying not only program steps and their parameters but also logical flow, including conditions. Kinesthetic teaching is utilized to obtain precise positions. However, its usage is minimized (to limit users' fatigue) only to get reference points, and then other necessary points are manipulated by graphical tools relative to the robot-originating points. This way, we also cope with the inaccuracy of AR registration in the real world. The framework also supports multiple users working simultaneously, which can be helpful in large-scale workplaces or during training.

Many rounds of internal testing were performed, focused both on the user interface as well as on the API of the framework, making sure that it is usable from both end-user and expert-user point of view. The role of expert users is mainly to develop the integration of devices and services and prepare necessary functionality on a task-appropriate level of abstraction. The end-users role is to program the task, but as the visual representation is compiled into a source code, expert users can get quickly involved even in this stage, when needed. Despite internal testing, the initial concept of the user interface was evaluated in a user study. After that, the interface design was changed, e.g., to allow controlling all UI elements by users' thumbs and thus reduce fatigue from holding a tablet in one hand. After the functionality was stabilized, an expert review was carried out, and the results are presented in this chapter. Its main outcomes are the necessity to implement an onboarding mode to support novice users, improve depth perception by providing additional cues, and simplify flow for adding actions.

In summary, when adopted by a system integrator and its customer, the framework allows efficient collaboration between professional (robot) programmers and end-users, who are domain experts. By allowing end-users to set up a workplace and create or adapt the program, the high flexibility needed for SMEs is achieved. Moreover, as SMEs can perform program modifications/adjustments on their own, expenses are reduced.

Part IV

CONCLUSIONS AND FUTURE RESEARCH

Based on my research presented in this thesis, I have proposed new approaches for the end-user robot programming task, using the projected SAR and mobile AR. Both advantages and limitations were discussed. The SAR approach has shown strong potential in table-top use cases, as it increases user awareness by presenting important information in the task space without any wearable device. On the other hand, its constraints in the 3D spatial parameters setting and visualization limit its usage. The mobile AR allowed the understandable manipulation and visualization of the 3D spatial information and the program flow. However, it requires the user to hold the mobile device in the user's hands.

The main contributions are the newly proposed interactive methods for virtual object selection in AR (O1)*, the definition of the individual robot action and their order in the program* (O2)*, and manipulation with the virtual objects in 3D space* (O3)*.*

11 CONCLUSIONS AND CONTRIBUTIONS

The end-user robot programming is an emerging field as it allows wider adoption of collaborative robots in SME. Compared to the traditional robot programming methods, it allows non-programmers to adapt the robotic programs or even create them from scratch. The cobots could therefore replace the shop-floor workers in repetitive and ergonomically challenging tasks and reduce the current problem of shop-floor workers' shortage.

The primary goal of the thesis was to enable end-users to program robots by ***bringing the interaction between humans and computers to the task space***. The thesis presents two different methods to achieve it. The first utilizes the SAR by projecting the user interface directly to the task space, and the second works with the AR on a tablet. Both of them benefit from the spatial nature of the robotic program and increase the robotic program comprehension through an understandable presentation of the program's spatial parameters.

The presented methods were evaluated with a total of more than 70 participants with respect to interface usability and user experience. The SAA method was also compared with the standard end-user robot programming tool, and it outperforms the existing solution in program comprehension while keeping the programming simplicity and usability at a similar level to the existing solution. The fully functional end-user robot programming tool was introduced, utilizing the SAA method, which is currently being deployed in the industry.

11.1 CONCLUSIONS AND RESEARCH OBJECTIVES

Four research objectives were stated to support the research statement. This section reviews the outcomes related to each objective and concludes the outcomes.

O1 – Define visualization and interaction methods suitable for such interfaces

In this thesis, I presented two main visualization approaches: the projected SAR and the AR on a mobile device. The SAR utilizes the projection of the user interface on the workbench (see Chapter 5), i.e., in the robot's and user's task space. It supports the user with information about the created program concerning the objects presented on the table.

The mobile AR visualize all the important information about the program and system state superimposed into the camera image on the mobile device's screen (see Chapter 7). The initial idea was transformed into the SAA method, which combines spatial anchors with user-defined actions for both visualization end execution purposes (see Chapter 8). The SAA method was utilized in a fully functional end-user robot programming tool and iteratively tested with several potential users (see Chapter 10).

Several interaction methods were examined for both visualization approaches. After initial experiments with gestural controls for the SAR in Section 3.3, the touch-enabled surface was selected as the primary interaction modality, in combination with direct robot manipulation for specification of selected 3D spatial information in Chapter 5.

For the mobile AR, the touch control was natural, in combination with the physical movement of the device (see Chapter 7). In the Chapter 9, the various interaction methods for object selection were examined and initially evaluated.

O2 – **Design a method for defining program flow**

The definition of the program flow was thoroughly investigated in Chapters 7, 8 and 10. The proposed method is based on so-called SAA, where the action representing the robot's actions or instructions are spatially anchored to the real place where the robot executes them. The programmer combines these individual actions into the form of an acyclic directed graph, which determines the program flow. The flow is visualized to the user as a series of connected 3D virtual objects representing the actions.

O3 – **Design a method for defining and visualizing the program's spatial parameters**

One of the most crucial parts of this work was enabling the end-user to define the program's spatial parameters properly. In the Chapter 5, the spatial parameter's setting using the projected user interface was presented. The programmer could use the touch gestures to manipulate various 2D widgets projected on the workbench to set parameters like the object's place position or polygonial area for object picking. The 3D spatial parameters could be set using direct robot manipulation, such as the object's position in the gravity feeder.

To further extend the possibilities of 3D spatial parameters definition, another set of 2D and 3D widgets for precise positioning of virtual objects was proposed in Chapters 8 and 10. By utilization of the hierarchical nature of the proposed Spatial Anchors in combination with precise navigation of the robotic arm, an implicit imprecision of the mobile AR is suppressed, and the programmer is enabled to define precise positions and trajectories.

O4 – **Evaluate the proposed method with non-programmers, concerning usability, mental workload, and user experience**

The proposed methods and user interfaces were thoroughly evaluated with potential users of such tools. Most of them had no experience with programming or AR. The experiment, presented in Chapter 5 has shown a strong potential of the projected SAR in the material handling scenarios and the high usability of the proposed method. The participants successfully set the program's spatial parameters and collaborated with the robot on the assembly task.

The experiment in Chapter 7 revealed the high usability of mobile AR in the visual inspection task. After a short training, the participants successfully created a robotic program using the prototype of the mobile interface. In the Chapter 8 a fully-functional tool was evaluated with 12 participants, showing a solid potential of the method and corresponding programming tool for spatial robotic tasks. In addition, the proposed interface was compared with the existing standard programming tool and showed better program readability while maintaining similar usability and time consumption. Several small experiments were conducted with the fully functional prototype. All of them are listed in Chapter 10.

11.2 FUTURE RESEARCH DIRECTIONS

The thesis so far presented up-to-date contributions. This section describes possible future research directions to build on these contributions.

11.2.1 *Simple and understandable definition of conditional behavior*

The conditional execution is an inevitable and crucial part of most programs, including robotic ones. Nevertheless, it is a non-trivial problem for the end-users using most visual programming methods. The definition of conditions and their connection to the other parts of the program (such as results of previously executed actions or inputs from sensors) could be tricky, especially for non-skilled operators.

The foundations of this research direction have already been laid in the paper [67], examining the basic definition of conditional behavior. However, it needs to be investigated more thoroughly to find an easily understandable and usable method for the condition definition by the end-users.

[1] Reprinted from `https://response.jp/article/2015/11/24/264810.html`

11.2.2 *Error indication and recoveries in flow-chart-like program representation*

As was revealed in my previous research, the end-users using our robot programming interfaces struggle to identify existing problems and errors in their programs. This problem is an exciting research direction, which is resolved satisfactorily in the traditional programming methods but remains a considerable problem in spatial AR programming tools.

Another problem is a error recovery, i.e. equivalent of the *exceptions* and *try – catch* blocks in traditional programming languages. It is also an interesting research direction I plan to investigate further.

11.2.3 *Haptic feedback for improvement of mobile AR*

Usage of haptic feedback for the mobile AR could be a possible game-changer, as it could significantly improve the usability of such interfaces. There are several prototypes of touch screen devices with haptic or tactile feedback, such as the one on the Fig. 11.1. The feedback could help the user to find the head-up user interface elements with their fingers without the need to fixate their sight on them.

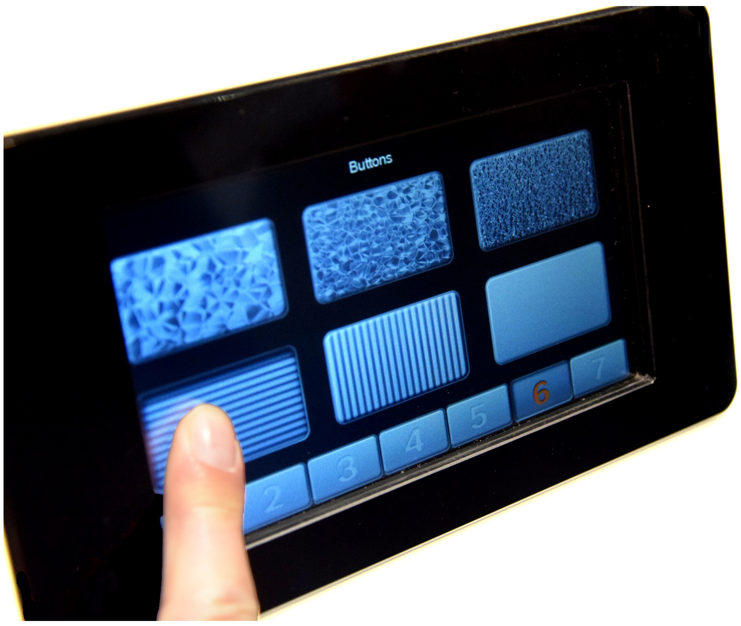


Figure 11.1: The touch screen device is equipped with a haptic feedback[1]. The 2D buttons provide feedback to the user, who can feel the button's texture with the tip of his finger.

BIBLIOGRAPHY

[1] Mechanical assembly assistance using marker-less augmented reality system. *Assembl. Autom.*, 38(1):77–87, feb 2018. ISSN 0144-5154. doi: 10.1108/AA-11-2016-152. URL https://doi.org/10.1108/AA-11-2016-152.

[2] Robotics ABB. Technical reference manual: Rapid instructions, functions and data types. *ABB Robotics*, 2014.

[3] Mustufa Haider Abidi, Abdulrahman Al-Ahmari, Ali Ahmad, Wadea Ameen, and Hisham Alkhalefah. Assessment of virtual reality-based manufacturing assembly training system. *The International Journal of Advanced Manufacturing Technology*, 105(9):3743–3759, Dec 2019. ISSN 1433-3015. doi: 10.1007/s00170-019-03801-3. URL https://doi.org/10.1007/s00170-019-03801-3.

[4] A. Adriaensen, F. Costantino, G. Di Gravio, and R. Patriarca. Teaming with industrial cobots: A socio-technical perspective on safety analysis. *Human Factors and Ergonomics in Manufacturing & Service Industries*, 32(2):173–198, 2022. doi: https://doi.org/10.1002/hfm.20939. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/hfm.20939.

[5] Salvador S. Agati, Rudieri D. Bauer, Marcelo da S. Hounsell, and Aleksander S. Paterno. Augmented reality for manual assembly in industry 4.0: Gathering guidelines. In *2020 22nd Symposium on Virtual and Augmented Reality (SVR)*, pages 179–188, 2020. doi: 10.1109/SVR51698.2020.00039.

[6] Gopika Ajaykumar and Chien-Ming Huang. User needs and design opportunities in end-user robot programming. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 93–95, 2020.

[7] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. A survey on end-user robot programming. *ACM Comput. Surv.*, 54(8), oct 2021. ISSN 0360-0300. doi: 10.1145/3466819. URL https://doi.org/10.1145/3466819.

[8] Jacopo Aleotti, Giorgio Micconi, and Stefano Caselli. Object interaction and task programming by demonstration in visuo-haptic augmented reality. *Multimedia Systems*, 22(6):675–691, Nov 2016. ISSN 1432-1882. doi: 10.1007/s00530-015-0488-z. URL https://doi.org/10.1007/s00530-015-0488-z.

[9] Sonya Alexandrova, Maya Cakmak, Kaijen Hsiao, and Leila Takayama. Robot programming by demonstration with interactive action visualizations. In *Robotics: science and systems*, pages 48–56. Citeseer, 2014.

[10] Ferran Argelaguet and Carlos Andujar. Efficient 3d pointing selection in cluttered virtual environments. *IEEE Computer Graphics and Applications*, 29 (6):34–43, 2009.

[11] Yusuf Aydin, Doganay Sirintuna, and Cagatay Basdogan. Towards collaborative drilling with a cobot using admittance controller. *Transactions of the Institute of Measurement and Control*, 43(8):1760–1773, 2021. doi: 10.1177/0142331220934643. URL https://doi.org/10.1177/0142331220934643.

[12] Daniel Bambušek, Zdeněk Materna, Michal Kapinus, and Vitězslav Beran. Handheld augmented reality: Overcoming reachability limitations by enabling temporal switching to virtual reality. In *Companion of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, 2022. Accepted.

[13] Daniel Bambuŝek, Zdeněk Materna, Michal Kapinus, Vítězslav Beran, and Pavel Smrž. Combining interactive spatial augmented reality with head-mounted display for end-user collaborative robot programming. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8, 2019. doi: 10.1109/RO-MAN46459.2019.8956315.

[14] E.I. Barakova, J.C.C. Gillesen, B.E.B.M. Huskens, and T. Lourens. End-user programming architecture facilitates the uptake of robots in social therapies. *Robotics and Autonomous Systems*, 61(7):704–713, 2013. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2012.08.001. URL https://www.sciencedirect.com/science/article/pii/S0921889012001182. Collective and Social Autonomous Robots.

[15] Esther Z Barsom, Maurits Graafland, and Marlies P Schijven. Systematic review on the effectiveness of augmented reality applications in medical training. *Surgical endoscopy*, 30(10):4174–4183, 2016.

[16] Fahmi Bellalouna. New approach for industrial training using virtual reality technology. *Procedia CIRP*, 93:262–267, 2020. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2020.03.008. URL https://www.sciencedirect.com/science/article/pii/S2212827120305539. 53rd CIRP Conference on Manufacturing Systems 2020.

[17] Hrvoje Benko, Andrew D. Wilson, and Federico Zannier. Dyadic projected spatial augmented reality. In *Proceedings of the 27th Annual ACM Sym-*

posium on User Interface Software and Technology, UIST '14, page 645–655, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330695. doi: 10.1145/2642918.2647402. URL https://doi.org/10.1145/2642918.2647402.

[18] Sourabh Bharti and Alan McGibney. Corol: A reliable framework for computation offloading in collaborative robots. *IEEE Internet of Things Journal*, pages 1–1, 2022. doi: 10.1109/JIOT.2022.3155587.

[19] Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. *Robot Programming by Demonstration*, pages 1371–1394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-30301-5. doi: 10.1007/978-3-540-30301-5_60. URL https://doi.org/10.1007/978-3-540-30301-5_60.

[20] Eva Binder, Jonas Teuber, Felix Gaisbauer, Thomas Bär, Thies Pfeiffer, and Sven Wachsmuth. Combining simulation and augmented reality methods for enhanced worker assistance in manual assembly. 81:588–593, 01 2019. doi: 10.1016/j.procir.2019.03.160.

[21] Sebastian Blankemeyer, Rolf Wiemann, Lukas Posniak, Christoph Pregizer, and Annika Raatz. Intuitive robot programming using augmented reality. *Procedia CIRP*, 76:155–160, 01 2018. doi: 10.1016/j.procir.2018.02.028.

[22] Petr Bobák, Ladislav Cmolik, and Martin Cadik. Temporally stable boundary labeling for interactive and non-interactive dynamic scenes. *Computers & Graphics*, 91, 08 2020. doi: 10.1016/j.cag.2020.08.005.

[23] Michael Borish and Jamie Westfall. Additive and subtractive manufacturing augmented reality interface (asmari). In *2020 SoutheastCon*, pages 1–6, 2020. doi: 10.1109/SoutheastCon44009.2020.9249710.

[24] Giovanni Boschetti, Matteo Bottin, Maurizio Faccio, and Riccardo Minto. Multi-robot multi-operator collaborative assembly systems: a performance evaluation model. *Journal of Intelligent Manufacturing*, 32(5):1455–1470, Jun 2021. ISSN 1572-8145. doi: 10.1007/s10845-020-01714-7. URL https://doi.org/10.1007/s10845-020-01714-7.

[25] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., USA, 2004. ISBN 0201758679.

[26] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[27] Fabio Bruno, Loris Barbieri, E. Marino, Maurizio Muzzupappa, Luigi D'Oriano, and Biagio Colacino. An augmented reality tool to detect and annotate design variations in an industry 4.0 approach. *The International Journal of Advanced Manufacturing Technology*, 105:1–13, 11 2019. doi: 10.1007/s00170-019-04254-4.

[28] Elsa Bunz, Ravi Teja Chadalavada, Henrik Andreasson, Robert Krug, Maike Schindler, and Achim Lilienthal. Spatial augmented reality and eye tracking for evaluating human robot interaction. In *RO-MAN 2016 Workshop: Workshop on Communicating Intentions in Human-Robot Interaction, New York, USA, Aug 31, 2016*, 2016.

[29] Andrzej Burghardt, Dariusz Szybicki, Piotr Gierlak, Krzysztof Kurc, Paulina Pietruś, and Rafał Cygan. Programming of industrial robots using virtual reality and digital twins. *Applied Sciences*, 10(2), 2020. ISSN 2076-3417. doi: 10.3390/app10020486. URL https://www.mdpi.com/2076-3417/10/2/486.

[30] Jonathan Cacace, Riccardo Caccavale, Alberto Finzi, and Vincenzo Lippiello. Variable admittance control based on virtual fixtures for human-robot co-manipulation. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1569–1574, 2019. doi: 10.1109/SMC.2019.8914084.

[31] Amedeo Cesta, Andrea Orlandini, Giulio Bernardi, and Alessandro Umbrico. Towards a planning-based framework for symbiotic human-robot collaboration. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2016. doi: 10.1109/ETFA.2016.7733585.

[32] Yeonjoo Cha and Rohae Myung. Extended fitts' law for 3d pointing tasks using 3d target arrangements. *International Journal of Industrial Ergonomics*, 43(4):350–355, 2013.

[33] Sonia Mary Chacko and Vikram Kapila. An augmented reality interface for human-robot interaction in unconstrained environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3222–3228, 2019. doi: 10.1109/IROS40897.2019.8967973.

[34] Seongju Chang, Sungil Ham, Seungbum Kim, Dongjun Suh, and Hyunseok Kim. Ubi-floor: Design and pilot implementation of an interactive floor system. In *2010 Second International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 290–293, 2010. doi: 10.1109/IHMSC.2010.172.

[35] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *Robotics & Automation Magazine*, 19(1):18–19, 2012.

[36] Christine Connolly. Technology and applications of abb robotstudio. *Industrial Robot: An International Journal*, 2009.

[37] SiHao Deng, ZhenHua Cai, DanDan Fang, HanLin Liao, and Ghislain Montavon. Application of robot offline programming in thermal spraying. *Surface and Coatings Technology*, 206(19):3875–3882, 2012. ISSN 0257-8972. doi: https://doi.org/10.1016/j.surfcoat.2012.03.038. URL https://www.sciencedirect.com/science/article/pii/S0257897212002071.

[38] Abhiraj Deshpande and Inki Kim. The effects of augmented reality on improving spatial problem solving for object assembly. *Advanced Engineering Informatics*, 38:760–775, 2018. ISSN 1474-0346. doi: https://doi.org/10.1016/j.aei.2018.10.004. URL https://www.sciencedirect.com/science/article/pii/S1474034618300971.

[39] Barrett Ens, Fraser Anderson, Tovi Grossman, Michelle Annett, Pourang Irani, and George Fitzmaurice. Ivy: Exploring spatially situated visual programming for authoring and understanding intelligent environments. In *Proceedings of the 43rd Graphics Interface Conference*, GI '17, pages 156–162, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2017. Canadian Human-Computer Communications Society. ISBN 978-0-9947868-2-1. doi: 10.20380/GI2017.20. URL https://doi.org/10.20380/GI2017.20.

[40] Henrik Eschen, Tobias Kötter, Rebecca Rodeck, Martin Harnisch, and Thorsten Schüppstuhl. Augmented and virtual reality for inspection and maintenance processes in the aviation industry. *Procedia manufacturing*, 19:156–163, 2018.

[41] Tobias Feigl, Andreas Porada, Steve Steiner, Christoffer Löffler, Christopher Mutschler, and Michael 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,*, pages 307–318. INSTICC, SciTePress, 2020. ISBN 978-989-758-402-2. doi: 10.5220/0008989903070318.

[42] Paul M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[43] Daniela Fogli, Luigi Gargioni, Giovanni Guida, and Fabio Tampalini. A hybrid approach to user-oriented programming of collaborative robots. *Robotics and Computer-Integrated Manufacturing*, 73:102234, 2022.

[44] J. Fryman and B. Matthias. Safety of industrial robots: From conventional to collaborative applications. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–5, 2012.

[45] Markus Funk. Augmented reality at the workplace: a context-aware assistive system using in-situ projection. 2016.

[46] Yuxiang Gao and Chien-Ming Huang. Pati: A projection-based augmented table-top interface for robot programming. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19, pages 345–355, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6272-6. doi: 10.1145/3301275.3302326. URL http://doi.acm.org/10.1145/3301275.3302326.

[47] Sofia Garcia Fracaro, Philippe Chan, Timothy Gallagher, Yusra Tehreem, Ryo Toyoda, Kristel Bernaerts, Jarka Glassey, Thies Pfeiffer, Bert Slof, Sven Wachsmuth, and Michael Wilk. Towards design guidelines for virtual reality training for the chemical industry. *Education for Chemical Engineers*, 36:12–23, 2021. ISSN 1749-7728. doi: https://doi.org/10.1016/j.ece.2021.01.014. URL https://www.sciencedirect.com/science/article/pii/S1749772821000142.

[48] Natanael Magno Gomes, Felipe Nascimento Martins, José Lima, and Heinrich Wörtche. Reinforcement learning for collaborative robots pick-and-place applications: A case study. *Automation*, 3(1):223–241, 2022. ISSN 2673-4052. doi: 10.3390/automation3010011. URL https://www.mdpi.com/2673-4052/3/1/11.

[49] Rebecca A. Grier. How high is high? a meta-analysis of nasa-tlx global workload scores. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 59(1):1727–1731, 2015. doi: 10.1177/1541931215591373. URL https://doi.org/10.1177/1541931215591373.

[50] Christian Groth and Dominik Henrich. One-shot robot programming by demonstration using an online oriented particles simulation. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 154–160, 2014. doi: 10.1109/ROBIO.2014.7090323.

[51] Kelleher R. Guerin, Sebastian D. Riedel, Jonathan Bohren, and Gregory D. Hager. Adjutant: A framework for flexible human-machine collaborative systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1392–1399, 2014. doi: 10.1109/IROS.2014.6942739.

[52] Kelleher R Guerin, Colin Lea, Chris Paxton, and Gregory D Hager. A framework for end-user instruction of a robot assistant for manufacturing. In *ICRA*, pages 6167–6174. IEEE, 2015.

[53] Anuja Hariharan, Bindu Maharudrappa, and Artur Felic. Augmented reality experiences for the operator 4.0. In Holger Fischer and Steffen Hess, editors, *Mensch und Computer 2020 - Usability Professionals*, Bonn, 2020. Gesellschaft für Informatik e.V. und German UPA e.V. doi: 10.18420/muc2020-up-0422.

[54] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology*, 52:139–183, 1988.

[55] Vincent Havard, Benoit Jeanne, Marc Lacomblez, and David Baudry. Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations. *Production & Manufacturing Research*, 7(1): 472–489, 2019. doi: 10.1080/21693277.2019.1660283. URL https://doi.org/10.1080/21693277.2019.1660283.

[56] Valentin Heun, James Hobin, and Pattie Maes. Reality editor: Programming smarter objects. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 307–310, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2215-7. doi: 10.1145/2494091.2494185. URL http://doi.acm.org/10.1145/2494091.2494185.

[57] Radovan Holubek, Roman Ružarovský, and Daynier R. D. Sobrino. An innovative approach of industrial robot programming using virtual reality for the design of production systems layout. In Justyna Trojanowska, Olaf Ciszak, José Mendes Machado, and Ivan Pavlenko, editors, *Advances in Manufacturing II*, pages 223–235, Cham, 2019. Springer International Publishing. ISBN 978-3-030-18715-6.

[58] John Horst, Elena Messina, Jeremy Marvel, et al. Best practices for the integration of collaborative robots into workcells within small and medium-sized manufacturing operations. *National Institute of Standards and Technology Advanced Manufacturing Series*, pages 100–41, 2021.

[59] Chien-Ming Huang. Contextual programming of collaborative robots. In *International Conference on Human-Computer Interaction*, pages 321–338. Springer, 2020.

[60] Gaoping Huang, Pawan Rao, Meng-Han Wu, Xun Qian, Shimon Nof, Karthik Ramani, and Alexander Quinn. Vipo: Spatial-visual program-

ming with functions for robot-iot workflows. pages 1–13, 04 2020. doi: 10.1145/3313831.3376670.

[61] Haosheng Huang, Manuela Schmidt, and Georg Gartner. Spatial knowledge acquisition with mobile maps, augmented reality and voice in the context of gps-based pedestrian navigation: Results from a field test. *Cartography and Geographic Information Science*, 39(2):107–116, 2012. doi: 10.1559/15230406392107. URL https://doi.org/10.1559/15230406392107.

[62] Justin Huang and Maya Cakmak. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In *HRI*, pages 453–462. ACM, 2017.

[63] Ifr. Demystifying collaborative industrial robots. Position paper, International Federation of Robotics, 10 2018.

[64] Tudor B Ionescu and Sebastian Schlund. A participatory programming model for democratizing cobot technology in public and industrial fablabs. *Procedia CIRP*, 81:93–98, 2019.

[65] ISO 8373:2012. Robots and robotic devices — Vocabulary. Standard, International Organization for Standardization, Geneva, CH, March 2012.

[66] Ömür Kaya Kalkan, Şener Karabulut, and Gürhan Höke. Effect of virtual reality-based training on complex industrial assembly task performance. *Arabian Journal for Science and Engineering*, 46(12):12697–12708, Dec 2021. ISSN 2191-4281. doi: 10.1007/s13369-021-06138-w. URL https://doi.org/10.1007/s13369-021-06138-w.

[67] Michal Kapinus, Vítězslav Beran, Zdeněk Materna, and Daniel Bambušek. Spatially situated end-user robot programming in augmented reality. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–8, 2019. doi: 10.1109/RO-MAN46459.2019.8956336.

[68] Michal Kapinus, Zdeněk Materna, Daniel Bambušek, and Vitězslav Beran. End-user robot programming case study: Augmented reality vs. teach pendant. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '20, page 281–283, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370578. doi: 10.1145/3371382.3378266. URL https://doi.org/10.1145/3371382.3378266.

[69] Michal Kapinus, Daniel Bambušek, Zdeněk Materna, and Vitězslav Beran. Improved indirect virtual objects selection methods for cluttered augmented reality environments on mobile devices. In *Companion of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, 2022.

[70] Michal Kapinus, Vítězslav Beran, Zdeněk Materna, and Daniel Bambušek. Spatially anchored actions: Comprehensible end-user robot programming in augmented reality. *Virtual Reality*, 2022. Submitted.

[71] Michal Kapinus, Zdeněk Materna, Vítězslav Beran, and Daniel Bambušek. Arcor2: Framework for collaborative end-user management of industrial robotic workplaces using augmented reality. *Journal of Intelligent & Robotic Systems*, 2022. Submitted.

[72] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3), apr 2011. ISSN 0360-0300. doi: 10.1145/1922649.1922658. URL https://doi.org/10.1145/1922649.1922658.

[73] Andreas Kokkas and George-Christopher Vosniakos. An augmented reality approach to factory layout design embedding operation simulation. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 13(3):1061–1071, Sep 2019. ISSN 1955-2505. doi: 10.1007/s12008-019-00567-6. URL https://doi.org/10.1007/s12008-019-00567-6.

[74] Regis Kopper, Felipe Bacim, and Doug A. Bowman. Rapid and accurate 3d selection by progressive refinement. In *2011 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 67–74, 2011. doi: 10.1109/3DUI.2011.5759219.

[75] PanosE Kourouthanassis, Costas Boletsis, and George Lekakos. Demystifying the design of mobile augmented reality applications. *Multimedia Tools and Applications*, 74:1–22, 02 2013. doi: 10.1007/s11042-013-1710-7.

[76] Matheus Ladeira, Yassine Ouhammou, and Emmanuel Grolleau. Robmex: Ros-based modelling framework for end-users and experts. *Journal of Systems Architecture*, 117:102089, 2021.

[77] Eva Lampen, Jonas Teuber, Felix Gaisbauer, Thomas Bär, Thies Pfeiffer, and Sven Wachsmuth. Combining simulation and augmented reality methods for enhanced worker assistance in manual assembly. *Procedia CIRP*, 81:588–593, 2019. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2019.03.160. URL https://www.sciencedirect.com/science/article/pii/S2212827119304652. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

[78] F. Leutert, C. Herrmann, and K. Schilling. A spatial augmented reality system for intuitive display of robotic data. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 179–180, 2013.

[79] Pai-Chia Li and Chih-Hsing Chu. Augmented reality based robot path planning for programming by demonstration. 12 2016.

[80] Ying Siu Liang, Damien Pellier, Humbert Fiorino, and Sylvie Pesty. iropro: An interactive robot programming framework. *International Journal of Social Robotics*, pages 1–15, 2021.

[81] Stefan Lichiardopol, Nathan van de Wouw, and Henk Nijmeijer. Control scheme for human-robot co-manipulation of uncertain, time-varying loads. In *2009 American Control Conference*, pages 1485–1490, 2009. doi: 10.1109/ACC.2009.5160062.

[82] Haomin Liu, Mingyu Chen, Guofeng Zhang, Hujun Bao, and Yingze 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*, pages 1974–1982, 2018.

[83] Yueyue Liu, Zhijun Li, Huaping Liu, and Zhen Kan. Skill transfer learning for autonomous robots and human–robot cooperation: A survey. *Robotics and Autonomous Systems*, 128:103515, 2020. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2020.103515. URL https://www.sciencedirect.com/science/article/pii/S0921889019309972.

[84] G. Maeda, A. Maloo, M. Ewerton, R. Lioutikov, and J. Peters. Anticipative interaction primitives for human-robot collaboration. In *AAAI Fall Symposium Series. Shared Autonomy in Research and Practice*, pages 325–330, November 2016. URL https://aaai.org/ocs/index.php/FSS/FSS16/paper/view/14067.

[85] Stéphane Magnenat, Morderchai Ben-Ari, Severin Klinger, and Robert W Sumner. Enhancing robot programming with visual feedback and augmented reality. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 153–158. ACM, 2015.

[86] Zhanat Makhataeva and Huseyin Atakan Varol. Augmented reality for robotics: A review. *Robotics*, 9(2), 2020. ISSN 2218-6581. doi: 10.3390/robotics9020021. URL https://www.mdpi.com/2218-6581/9/2/21.

[87] Virginia Mamone, Fabrizio Cutolo, Sara Condino, and Vincenzo Ferrari. Projected augmented reality to guide manual precision tasks: An alternative to

head mounted displays. *IEEE Transactions on Human-Machine Systems*, 52(4): 567–577, 2022. doi: 10.1109/THMS.2021.3129715.

[88] Evgenia Manou, George-Christopher Vosniakos, and Elias Matsas. Off-line programming of an industrial robot in a virtual reality environment. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 13(2): 507–519, Jun 2019. ISSN 1955-2505. doi: 10.1007/s12008-018-0516-2. URL https://doi.org/10.1007/s12008-018-0516-2.

[89] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[90] C. Mateo, A. Brunete, E. Gambao, and M. Hernando. Hammer: An android based application for end-user industrial robot programming. In *2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, pages 1–6, Sep. 2014. doi: 10.1109/MESA.2014.6935597.

[91] Z. Materna, M. Kapinus, V. Beran, P. Smrž, and 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)*, pages 80–87, Aug 2018. doi: 10.1109/ROMAN.2018.8525662.

[92] Zdeněk Materna, Michal Kapinus, et al. Using persona, scenario, and use case to develop a human-robot augmented reality collaborative workspace. In *HRI*, pages 1–2. ACM, 2017. ISBN 978-1-4503-4885-0.

[93] Zdeněk Materna, Michal Kapinus, Michal Španěl, Vítězslav Beran, and Pavel Smrž. Simplified industrial robot programming: Effects of errors on multimodal interaction in woz experiment. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 200–205, 2016. doi: 10.1109/ROMAN.2016.7745111.

[94] Christoph Mayr-Dorn, Mario Winterer, Christian Salomon, Doris Hohensinger, and Rudolf 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)*, pages 5–12. IEEE, 2021.

[95] Annette Mossel, Benjamin Venditti, and Hannes Kaufmann. Drillsample: precise selection in dense handheld augmented reality environments. In

Proceedings of the Virtual Reality International Conference: Laval Virtual, pages 1–10, 2013.

[96] Stefanos Nikolaidis, Przemyslaw Lasota, Ramya Ramakrishnan, and Julie Shah. Improved human–robot team performance through cross-training, an approach inspired by human team training practices. *The International Journal of Robotics Research*, 34(14):1711–1730, 2015. doi: 10.1177/0278364915609673. URL https://doi.org/10.1177/0278364915609673.

[97] Petr Novák, Šimon Stoszek, and Jiří Vyskočil. Calibrating industrial robots with absolute position tracking system. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1187–1190, 2020. doi: 10.1109/ETFA46521.2020.9212169.

[98] Paweł Nowacki and Marek Woda. Capabilities of arcore and arkit platforms for ar/vr applications. In *International Conference on Dependability and Complex Systems*, pages 358–370. Springer, 2019.

[99] Manoch Numfu, Andreas Riel, and Frederic Noel. Virtual reality based digital chain for maintenance training. *Procedia CIRP*, 84:1069–1074, 2019. ISSN 2212-8271. doi: https://doi.org/10.1016/j.procir.2019.04.268. URL https://www.sciencedirect.com/science/article/pii/S2212827119309151. 29th CIRP Design Conference 2019, 08-10 May 2019, Póvoa de Varzim, Portgal.

[100] S.K. Ong, A.W.W. Yew, N.K. Thanigaivel, and A.Y.C. Nee. Augmented reality-assisted robot programming system for industrial applications. *Robotics and Computer-Integrated Manufacturing*, 61:101820, 2020. ISSN 0736-5845. doi: https://doi.org/10.1016/j.rcim.2019.101820. URL https://www.sciencedirect.com/science/article/pii/S0736584519300250.

[101] Eric M Orendt, Myriel Fichtner, and Dominik Henrich. Robot programming by non-experts: Intuitiveness and robustness of one-shot robot programming. In *RO-MAN*, pages 192–199. IEEE, 2016.

[102] M. Ostanin and A. Klimchik. Interactive robot programing using mixed reality. *IFAC-PapersOnLine*, 51(22):50 – 55, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.11.517. URL http://www.sciencedirect.com/science/article/pii/S2405896318332233. 12th IFAC Symposium on Robot Control SYROCO 2018.

[103] Yoon Jung Park, Yoonsik Yang, Seungho Chae, Inhwan Kim, and Tack-don Han. Designar: Portable projection-based ar system specialized in interior design. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2879–2884, 2017. doi: 10.1109/SMC.2017.8123064.

[104] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D Hager. Costar: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 564–571. IEEE, 2017.

[105] Patrick Perea, Denis Morand, and Laurence Nigay. Target expansion in context: the case of menu in handheld augmented reality. In *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 1–9, 2020.

[106] Alexander Perzylo, Nikhil Somani, Stefan Profanter, Ingmar Kessler, Markus Rickert, and Alois Knoll. Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In *2016 ieee/rsj international conference on intelligent robots and systems (iros)*, pages 2293–2300. IEEE, 2016.

[107] Jarkko Polvi, Takafumi Taketomi, Goshiro Yamamoto, Arindam Dey, Christian Sandor, and Hirokazu Kato. Slidar: A 3d positioning method for slam-based handheld augmented reality. *Computers & Graphics*, 55:33–43, 2016. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2015.10.013. URL https://www.sciencedirect.com/science/article/pii/S0097849315001806.

[108] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 79–80, 1996.

[109] Yuan Yuan Qian and Robert J Teather. The eyes don't have it: an empirical comparison of head-based and eye-based selection in virtual reality. In *Proceedings of the 5th Symposium on Spatial User Interaction*, pages 91–98, 2017.

[110] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009.

[111] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. F. Machiel Van der Loos, and E. Croft. Robot programming through augmented trajectories in augmented reality. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1838–1844, Oct 2018. doi: 10.1109/IROS.2018.8593700.

[112] Iulian Radu, Blair MacIntyre, and Stella Lourenco. Comparing children's crosshair and finger interactions in handheld augmented reality: Relationships between usability and child development. In *Proceedings of the The 15th International Conference on Interaction Design and Children*, pages 288–298, 2016.

[113] Universal Robot. The urscript programming language for e-series. *Universal Robot*, 2022.

[114] Robotiq. Collaborative robot. Buyer's guide. URL `https://blog.robotiq.com/hubfs/COBOT%20EBOOK%20FINAL.pdf`.

[115] Guilherme Boulhosa Rodamilans, Emília Villani, Luís Gonzaga Trabasso, Wesley Rodrigues de Oliveira, and Ricardo Suterio. A comparison of industrial robots interface: force guidance system and teach pendant operation. *Industrial Robot: An International Journal*, 43(5):552–562, 2016.

[116] E.M. Rogers. *Diffusion of innovations*. Free Press of Glencoe, 1962. URL `https://books.google.cz/books?id=zw0-AAAAIAAJ`.

[117] Juan Roldán Gómez, Elena Crespo, Andrés Martín-Barrio, Elena Peña-Tapia, and Antonio Barrientos. A training system for industry 4.0 operators in complex assemblies based on virtual reality and process mining. *Robotics and Computer-Integrated Manufacturing*, 59:305–316, 05 2019. doi: 10.1016/j.rcim.2019.05.004.

[118] Francisco J Romero-Ramirez, Rafael Muñoz-Salinas, and Rafael Medina-Carnicer. Speeded up detection of squared fiducial markers. *Image and vision Computing*, 76:38–47, 2018.

[119] Eric Rosen, David Whitney, Elizabeth Phillips, Gary Chien, James Tompkin, George Konidaris, and Stefanie Tellex. Communicating robot arm motion intent through mixed reality head-mounted displays. In Nancy M. Amato, Greg Hager, Shawna Thomas, and Miguel Torres-Torriti, editors, *Robotics Research*, pages 301–316, Cham, 2020. Springer International Publishing. ISBN 978-3-030-28619-4.

[120] Daniel Rubarth. The cobot puts the finishing touches. *IST International Surface Technology*, 15(1):54–57, Apr 2022. ISSN 2192-8703. doi: 10.1007/s35724-022-0521-7. URL `https://doi.org/10.1007/s35724-022-0521-7`.

[121] Ali Samini and Karljohan Lundin Palmerius. Wand-like interaction with a hand-held tablet device—a study on selection and pose manipulation techniques. *Information*, 10(4):152, 2019.

[122] Marc Ericson Santos, Jarkko Polvi, Takafumi Taketomi, Goshiro Yamamoto, Christian Sandor, and Hirokazu Kato. A usability scale for handheld augmented reality. 11 2014. doi: 10.1145/2671015.2671019.

[123] Jeff Sauro and James R. Lewis. *Quantifying the User Experience: Practical Statistics for User Research*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012. ISBN 9780123849687, 9780123849694.

[124] Tim Scargill, Jiasi Chen, and Maria Gorlatova. Here to stay: Measuring hologram stability in markerless smartphone augmented reality. *arXiv preprint arXiv:2109.14757*, 2021.

[125] Christina Schmidbauer, Titanilla Komenda, and Sebastian Schlund. Teaching cobots in learning factories–user and usability-driven implications. *Procedia manufacturing*, 45:398–404, 2020.

[126] Susanne Schmidt, Frank Steinicke, Andrew Irlitti, and Bruce H. Thomas. Floor-projected guidance cues for collaborative exploration of spatial augmented reality setups. In *Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces*, ISS '18, page 279–289, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356947. doi: 10.1145/3279778.3279806. URL https://doi.org/10.1145/3279778.3279806.

[127] Jan Schmitt, Andreas Hillenbrand, Philipp Kranz, and Tobias Kaupp. Assisted human-robot-interaction for industrial assembly: Application of spatial augmented reality (sar) for collaborative assembly tasks. In *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '21 Companion, page 52–56, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382908. doi: 10.1145/3434074.3447127. URL https://doi.org/10.1145/3434074.3447127.

[128] Casper Schou, Rasmus Skovgaard Andersen, Dimitrios Chrysostomou, Simon Bøgh, and Ole Madsen. Skill-based instruction of collaborative robots in industrial settings. *Robotics and Computer-Integrated Manufacturing*, 53:72–80, 2018. ISSN 0736-5845. doi: https://doi.org/10.1016/j.rcim.2018.03.008. URL https://www.sciencedirect.com/science/article/pii/S0736584516301910.

[129] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. Construction of a benchmark for the user experience questionnaire (ueq). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4:40–44, 06 2017. doi: 10.9781/ijimai.2017.445.

[130] Yasaman S. Sefidgar, Prerna Agarwal, and Maya Cakmak. Situated tangible robot programming. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, page 473–482, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450343367. doi: 10.1145/2909824.3020240. URL https://doi.org/10.1145/2909824.3020240.

[131] Yasaman S Sefidgar, Thomas Weng, Heather Harvey, Sarah Elliott, and Maya Cakmak. Robotist: Interactive situated tangible robot programming. In *Proceedings of the Symposium on Spatial User Interaction*, pages 141–149, 2018.

[132] Margarida Silva, João Pedro Dias, André Restivo, and Hugo Sereno Ferreira. A review on visual programming for distributed computation in iot. In Maciej Paszynski, Dieter Kranzlmüller, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot, editors, *Computational Science – ICCS 2021*, pages 443–457, Cham, 2021. Springer International Publishing. ISBN 978-3-030-77970-2.

[133] Maximilian Speicher, Brian D. Hall, and Michael Nebeling. What is mixed reality? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300767. URL https://doi.org/10.1145/3290605.3300767.

[134] Susanne Stadler, Kevin Kain, Manuel Giuliani, Nicole Mirnig, Gerald Stollnberger, and Manfred Tscheligi. Augmented reality for industrial robot programmers: Workload analysis for task-based, augmented reality-supported robot control. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*, pages 179–184. IEEE, 2016.

[135] Susanne Stadler, Nicole Mirnig, Manuel Giuliani, Manfred Tscheligi, Zdenek Materna, and Michal Kapinus. Industrial human-robot interaction: Creating personas for augmented reality supported robot control and teaching. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, page 291–292, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348850. doi: 10.1145/3029798.3038365. URL https://doi.org/10.1145/3029798.3038365.

[136] Maj Stenmark, Mathias Haage, and Elin Anna Topp. Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation. In *HRI*, pages 463–472. ACM, 2017.

[137] Craig J. Sutherland and Bruce A. MacDonald. Naoblocks: A case study of developing a children's robot programming environment. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 431–436, 2018. doi: 10.1109/URAI.2018.8441843.

[138] Ryo Suzuki, Adnan Karim, Tian Xia, Hooman Hedayati, and Nicolai 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*, pages 1–33, 2022.

[139] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11, 2017.

[140] Markus Tatzgern, Valeria Orso, Denis Kalkofen, Giulio Jacucci, Luciano Gamberini, and Dieter Schmalstieg. Adaptive information density for augmented reality displays. In *2016 IEEE Virtual Reality (VR)*, pages 83–92, 2016. doi: 10.1109/VR.2016.7504691.

[141] Robert J Teather and Wolfgang Stuerzlinger. Pointing at 3d targets in a stereo head-tracked virtual environment. In *2011 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 87–94. IEEE, 2011.

[142] Takahiro Terashima and Osamu Hasegawa. A visual-slam for first person vision and mobile robots. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 73–76. IEEE, 2017.

[143] Yong Joon Thoo, Jérémy Maceiras, Philip Abbet, Mattia Racca, Hakan Girgin, and Sylvain Calinon. Online and offline robot programming via augmented reality workspaces. *arXiv preprint arXiv:2107.01884*, 2021.

[144] Eduardo Veas and Ernst Kruijff. Vesp'r: design and evaluation of a handheld ar device. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 43–52, 2008. doi: 10.1109/ISMAR.2008.4637322.

[145] Aida Vidal-Balea, Oscar Blanco-Novoa, Paula Fraga-Lamas, Miguel Vilar-Montesinos, and Tiago M. Fernández-Caramés. Creating collaborative augmented reality experiences for industry 4.0 training and assistance applications: Performance evaluation in the shipyard of the future. *Applied Sciences*, 10(24), 2020. ISSN 2076-3417. doi: 10.3390/app10249073. URL https://www.mdpi.com/2076-3417/10/24/9073.

[146] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248 – 266, 2018. ISSN 0957-4158. doi: https://doi.org/10.1016/j.mechatronics.2018.02.009. URL http://www.sciencedirect.com/science/article/pii/S0957415818300321.

[147] Thomas Vincent, Laurence Nigay, and Takeshi Kurata. Handheld augmented reality: Effect of registration jitter on cursor-based pointing techniques. In *Proceedings of the 25th Conference on l'Interaction Homme-Machine*, pages 1–6, 2013.

[148] Kenneth Walsh and Suzanne Pawlowski. Virtual reality: A technology in need of is research. *Communications of the AIS*, 8, 03 2002. doi: 10.17705/1CAIS.00820.

[149] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C Shepherd, and Diana Franklin. Evaluating coblox: A comparative

study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 366. ACM, 2018.

[150] Astrid Weiss, Andreas Huber, Jürgen Minichberger, and Markus Ikeda. First application of robot teaching in an existing industry 4.0 environment: Does it really work? *Societies*, 6(3):20, 2016.

[151] Stefan Werrlich, Kai Nitsche, and Gunther Notni. Demand analysis for an augmented reality based assembly training. In *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 416–422, 2017.

[152] Isabell Wohlgenannt, Alexander Simons, and Stefan Stieglitz. Virtual reality. *Business & Information Systems Engineering*, 62(5):455–461, Oct 2020. ISSN 1867-0202. doi: 10.1007/s12599-020-00658-9. URL https://doi.org/10.1007/s12599-020-00658-9.

[153] Daniela Wurhofer, Verena Fuchsberger, et al. Insights from user experience research in the factory: What to consider in interaction design. In *Human Work Interaction Design. Work Analysis and Interaction Design Methods for Pervasive and Smart Workplaces*, pages 39–56. Springer, 2015.

[154] Robert Xiao, Scott Hudson, and Chris Harrison. Supporting responsive cohabitation between virtual interfaces and physical objects on everyday surfaces. *HCI*, 1(1):12, 2017.

[155] Jianghao Xiong, En-Lin Hsiang, Ziqian He, Tao Zhan, and Shin-Tson Wu. Augmented reality and virtual reality displays: emerging technologies and future perspectives. *Light: Science & Applications*, 10(1):216, Oct 2021. ISSN 2047-7538. doi: 10.1038/s41377-021-00658-8. URL https://doi.org/10.1038/s41377-021-00658-8.

[156] Enes Yigitbas, Ivan Jovanovikj, and Gregor Engels. Simplifying robot programming using augmented reality and end-user development. *arXiv preprint arXiv:2106.07944*, 2021.

[157] Jibin Yin, Chengyao Fu, Xiangliang Zhang, and Tao Liu. Precise target selection techniques in handheld augmented reality interfaces. *IEEE Access*, 7: 17663–17674, 2019.

[158] Samir Yitzhak Gadre, Eric Rosen, Gary Chien, Elizabeth Phillips, Stefanie Tellex, and George Konidaris. End-user robot programming using mixed reality. 10 2018. doi: 10.13140/RG.2.2.18572.97926.

[159] Przemysław Zawadzki, Krzysztof Żywicki, Paweł Buń, and Filip Górski. Employee training in an intelligent factory using virtual reality. *IEEE Access*, 8: 135110–135117, 2020. doi: 10.1109/ACCESS.2020.3010439.

[160] Yong Zhou, Yiqun Peng, Weidong Li, and Duc Truong Pham. Stackelberg model-based human-robot collaboration in removing screws for product remanufacturing. *Robotics and Computer-Integrated Manufacturing*, 77:102370, 2022. ISSN 0736-5845. doi: https://doi.org/10.1016/j.rcim.2022.102370. URL `https://www.sciencedirect.com/science/article/pii/S0736584522000576`.

[161] Kamil Židek, Ján Piteľ, Milan Adámek, Peter Lazorík, and Alexander Hošovský. Digital twin of experimental smart manufacturing assembly system for industry 4.0 concept. *Sustainability*, 12(9), 2020. ISSN 2071-1050. doi: 10.3390/su12093658. URL `https://www.mdpi.com/2071-1050/12/9/3658`.

Part V

APPENDIX

Some outcomes of my PhD study, which are important for the author and the inquisitive reader, but not for the technical text above, are presented in the following pages.

A PAPERS NOT INCLUDED IN THIS THESIS

This appendix presents the papers I have participated in but were not included in this thesis. During my stay at the University of Salzburg, I have participated in two papers with my colleagues from the Center for Human-Computer Interaction of said university. In the papers, namely *Industrial Human-Robot Interaction: Creating Personas for Augmented Reality supported Robot Control and Teaching* [135] and *Using Persona, Scenario, and Use Case to Develop a Human-Robot Augmented Reality Collaborative Workspace* [92], we have initially investigated the problems which the presented theses deal with. The papers focus mainly on the application of Human-Computer Interaction (HCI) approaches in the HRI field and presents the proposed SAR interface for the first time.

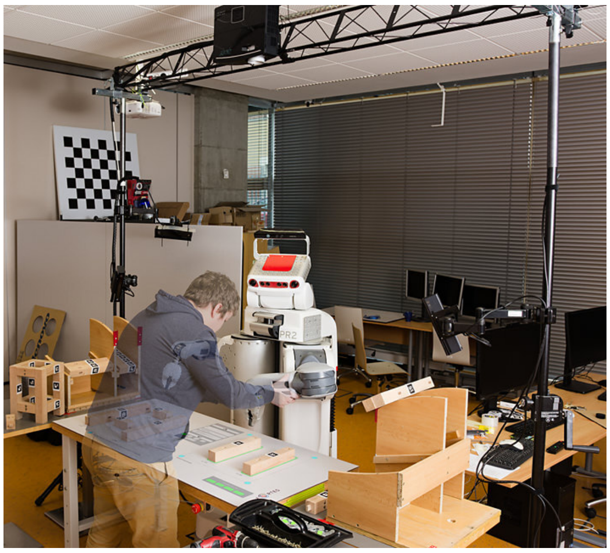
I have also participated in my colleague's research, summarized in the paper *Combining Interactive Spatial Augmented Reality with Head-Mounted Display for End-User Collaborative Robot Programming* [13]. It was the preliminary insight into the three-dimensional AR. The SAR interface, presented in the Chapter 5 was combined with the newly proposed HMD-based support interface, mainly for the definition of 3D spatial information. My contribution to this paper was mainly the collaboration on the design of the proposed HMD AR interface and participation in the design end evaluation of the user study.

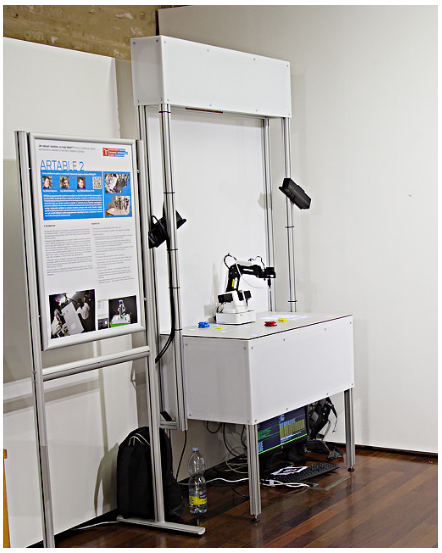
When we moved on to the mobile AR, we also investigated the pros and cons of the non-immersive VR on mobile devices. The outcomes of our research were summarized in the paper *Overcoming Reachability Limitations by Enabling Temporal Switch to Virtual Reality* [12]. I have participated in the UI design and partially in the experiment design and evaluation.

B DEVELOPED SOFTWARE

Alongside my research, several robotic work cells (see Fig. B.1 and Fig. B.2) involving various robots, different HW (projectors, cameras, and others), and using software we developed were deployed. Software used in these setups, which I have contributed in, involves primarily but not exclusively:

- Methods for the mutual calibration of cameras, robots, projectors, and touch-enabled surfaces.
- Object detection using fiducial markers and color features.
- Collision objects manager.
- Projected user interface.
- Awareness interfaces using light and sound.
- Application interfaces for various robots.
- Mobile interface for robot programming, based on proposed SAA method.


(a) The first generation of interactive table with projected SAR interface.



(b) The second generation of interactive table with projected SAR interface.

Figure B.1: The two ARTable setups using the PR2 robot (a) and the Dobot Magician (b).

The robots used and integrated into various work cells during the work on my thesis are presented in the following list:

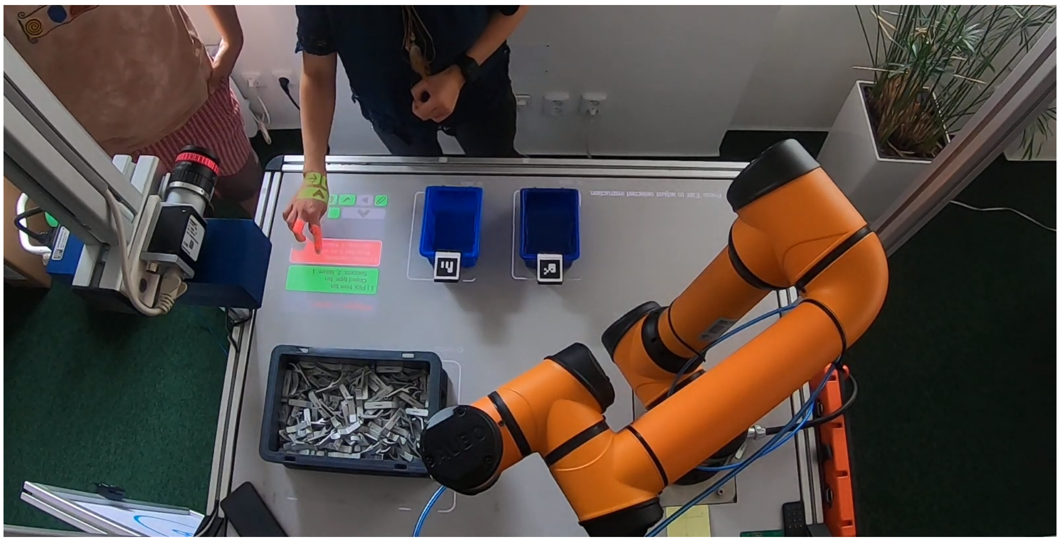


Figure B.2: The projected user interface with integrated robot Aubo i5.

- Dobot Magician
- Dobot M1
- Niryo One
- PR2
- Aubo i5
- ABB YuMi
- FANUC LRMate 200iD/7L
- Fanuc CRX

Apart from the original ARTable/ARCOR system, the second generation of programming system called ARCOR2 was created, involving primarily:

- **AREditor** - 3D scene and program editor using AR on a mobile device, together with desktop version mainly for experimental purposes
- **ARServer** - server unit operating as a provider of services for AREditor, responsible for synchronizing multiple connected AREditors, and other interfaces.
- Other services, responsible for conversion of the program described by connected actions to Python script and vice versa, executions of such scripts, and other supportive services.

At the time of finishing this thesis, the initial prototype described in Chapter 10 was being transformed into a commercial product and was initially deployed in the industry.

All of the created software is open-source and publicly available on web of our research group Robo@FIT[1] and on GitHub[2].

1 `https://www.fit.vut.cz/research/group/robo/`

2 `https://github.com/robofit/arcor`, `https://github.com/robofit/arcor2` and `https://github.com/robofit/arcor2_areditor`

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". It is available for LaTeX via CTAN as `classicthesis`.