Enhancing Robot Programming through Digital Twin and Augmented Reality

Enes Yigitbas
Paderborn University
enes@mail.upb.de

Gregor Engels
Paderborn University
engels@upb.de

Abstract

Nowadays, robots are widespread across diverse application contexts. However, robot programming is a cumbersome and error-prone task that requires a high domain and programming expertise. To simplify the process of robot programming, we combine Augmented Reality (AR) with the concept of Digital Twin (DT). By combining them, the robot system can be simulated through a digital equivalent representation while the real environment is extended with useful virtual artifacts. To enable users to work in the robot space, reducing the amount of mentally taxing coordinate space conversions, we have developed the DT- and AR-based robot programming framework, called DART. DART supports users to program a robot through interactive gestures, offers AR in-place program simulation, and direct building of finished programs to the real robot. We evaluated our AR-based programming approach regarding usability compared to a web-based robot programming approach. The evaluation showed that our approach is more usable than the conventional method and has the potential to enrich and ease current robot programming processes.

Keywords: digital twin, augmented reality, robot programming

1. Introduction

Nowadays, robots are an important tool that are used in different aspects of our lives, performing repetitive, delicate, or dangerous tasks. Depending on the context of use, robots become increasingly specialized in physical aspects as well as in their programming. However, robot programming is complicated and time-consuming as there is a lot of (mental) translation between different coordinate spaces (e.g. robot, mental, digital space) that needs to be addressed. Furthermore, the effect of robot programs is not always easy to predict with conventional robot programming interfaces showing robot simulations on a 2D screen decoupled from the real-world scenario where the robot should perform its tasks.

To tackle these challenges and to overcome the drawbacks of conventional robot programming interfaces, we present a Digital Twin (DT)- and Augmented Reality (AR)-based robot programming framework, called DART. It supports end-users without significant robot- or general programming experience to program robots by themselves. For this purpose, DART makes use of a DT representation of a real robot in the form of an equivalent virtual robot. This DT robot is used as an AR projection in real-world scenes. Based on a Blockly-based[1] AR programming interface, the users create their robot commands on the virtual programming interface. Furthermore, the programming and editing of the virtual robot are enabled through natural interaction gestures like setting waypoints and drawings for specifying the robot movements. The so-created robot program can be simulated through the DT of the robot as DART offers an AR in-place program simulation where the virtual robot executes the specified program commands. Furthermore, DART supports the direct building of finished programs for the real robot. Once the user is satisfied with the created program and its simulation, the robot program can be deployed to the real robot that executes the created robot program. Based on this programming style, create-simulate-evaluate cycles are possible that support

[1] https://developers.google.com/blockly

URI: https://hdl.handle.net/10125/103452
978-0-9981331-6-4

the agile development of complex robot programs. This is mainly possible due to the combination of a digital twin representation and its usage in augmented reality that is both combined in our solution approach that aims to ease the process of robot programming. We evaluated the usability of DART in terms of efficiency, effectiveness, user satisfaction, and cognitive workload and compared it to a conventional web-based robot programming approach without AR and DT. The evaluation showed that our approach is more usable than the conventional method, especially in terms of user satisfaction and cognitive workload, and has the potential to enrich and ease current robot programming processes.

The rest of the paper is structured as follows: In Section 2, we present existing work. Details on the conception and implementation of DART are given in Section 3. In Section 4, we present the evaluation and its main results. Section 5 concludes the paper and gives an outlook for future work.

2. Related Work

Augmented Reality (AR) and Virtual Reality (VR) have been a topic of intense research in the last decades. In the past few years, massive advances in affordable consumer hardware and accessible software frameworks are now bringing AR and VR to the masses. While VR interfaces support the interaction in an immersive computer-generated 3D world, AR enables the augmentation of real-world physical objects with virtual elements. In conjunction with these technologies, for the last two decades, researchers have been working on the development of Digital Twin models. A Digital Twin maps a physical object or process with a digital environment that enables simulation, education, prediction, and optimization and helps in product design and simulating manufacturing systems, and processes (Garg et al., 2021). Robotic applications are a specific area of the Digital Twin domain, focusing on the ease of programming (Zhang et al., 2019). In the following, we draw on recent research for AR- and Digital Twin-based robot programming approaches.

2.1. Augmented Reality-based Approaches

In the following, we briefly present and discuss related approaches that follow the idea of augmented reality-assisted robot programming.

Shepherd et al., 2019 used a video-based AR approach for robot programming. A block-based Integrated Development Environment (IDE) was embedded onto the screen. The block-based IDE is a CoBlox derivate. CoBlox is designed for offline robot programming and normally features a robot and environment simulation (Weintrop et al., 2017, Shepherd et al., 2018, Weintrop et al., 2018). In Shepherd et al., 2019 it was also noted, that programming with controllers as well as having to move the real robot with hands is cumbersome.

Another approach is to use hands for both interactions. This approach was used in Gadre et al., 2019 where a Mixed Reality (MR) interface was proposed for creating and editing waypoints. The created waypoints could be grouped, the resulting action previewed, and the resulting program executed on the real robot. The solution was tested against a monitor interface in a usability study using Microsoft HoloLens.

Thanigaivel et al., 2019 proposed an AR-assisted robot programming system that allows users to program pick-and-place as well as welding tasks by demonstrating the task/path. These paths can be selected by showing the full path, showing the start and endpoints, or selecting features based on Computer-Aided Design (CAD), e.g., edges. The robot motion was simulated and augmented in AR.

Rosen et al., 2019 proposed an MR interface for programming robots. Here a manipulation of the movement starting point and goal, as well as end-effector orientation, via hand interaction, is possible. The AR device used is a Microsoft HoloLens. MoveIt is the motion planning tool used to calculate a path between the two aforementioned points (start point and endpoint).

In summary, the idea of AR-assisted robot programming is an established field of research and there are existing approaches that make use of AR technology to ease the complex task of robot programming. However, most of the existing approaches are focusing on specific 3D objects and virtual representations of robots neglecting the integration of digital twins in an AR-assisted robot programming environment in a systematic way. To fill this gap, our solution approach aims to combine the concept of the digital twin with AR technology. For this purpose, we will show how different DTs of a robot system can be integrated and used in an AR-assisted robot programming environment to reduce the complexity of robot programming.

2.2. Digital Twin-based Approaches

The digital twin is an idea related to the concept of a virtual, digital equivalent of a physical product. The digital twin is used to reproduce a real, physically existing environment, process, or single object. Its

role is to map the main features of the physical object or process enabling simulation, prediction and optimization in the areas of system servicing (Olabi et al., 2012), product design and manufacturing systems (Burghardt et al., 2020), and processes (J. W. Hart et al., 2021). The main features of the digital twin, such as its form, upgradeability, the degree of precision with which it maps reality, and many others, depend on the purpose it serves (Muszyńska et al., 2019).

In previous work, the concept of the digital twin has been used in several domains such as manufacturing (Tao et al., 2018) or assistance systems (Josifovska et al., 2019). Utilizing a combination of the concept of digital twin and VR interfaces has been shown successfully in previous works, especially for human-robot interaction (HRI). Whitney et al., 2018 suggest that VR interfaces can allow non-expert users to control robots, and present a controlling mechanism where the user's hands are directly mapped to the robot's hands. To support controllability of robot systems, Lipton et al., 2018 present a VR interface that simulates a control room (VRCR), where the user sees the world from the robot's eyes and uses control orbs to move its arms. Another approach was presented by Burghardt et al., 2020, where a VR interface including the digital twin of a robotic station, tracks user interaction with a real object for programming the robot. Similarly, the approach by Theofanidis et al., 2017 enables controlling a robot in VR, by allowing the user to define the trajectory that the robot arm should follow by detecting hand gestures of the user.

In summary, most of the existing digital twin-based approaches for supporting robot programming rely on virtual reality technology. While such approaches support the simulation of robot systems in a virtual environment, we believe that AR technology is better suited to simulate a robot system in a real-world setup as collisions and other environmental parameters can be taken into consideration. Therefore, our solution approach focuses on the usage of AR technology and its combination with digital twin models of a robot to leverage an easy-to-use robot programming interface.

3. Conceptual Solution and Implementation

To ease the process of robot programming by combining the concept of digital twin and augmented reality, we have created the DART framework. The architectural overview of our DART framework is shown in Figure 1. It is mainly based on a 4-layer architecture consisting of the following layers: *Robotics Layer*, *Translation Layer*, *Programming Layer*, and *UI Layer*. In the following, each layer and its implementation will be described in more detail.

The *Robotics Layer* contains the main entities that have to be programmed. In our case, this layer consists of the main components *Lego NXT* and *Dobot* as we have shown the functionality of the DART framework based on these exemplary robot systems. Each robot component is divided up into two subcomponents: one subcomponent characterizing the real robot and the other one characterizing the virtual robot that is the digital twin model of the real robot system. We have chosen the *Lego NXT*[2] and *Dobot*[3] systems to have one example for a mobile and a static industrial robot arm, respectively. For both virtual robot representations, the *Digital-Twin NXT* and the *Digital-Twin Dobot*, we have created a 3D model based on Blender[4]. These 3D models were imported into our project so that the digital twin representation was accessible by the the *Translation Layer*.

In the *Translation Layer*, we have an *Execution* engine that is responsible for steering the real robots and controlling the simulation of the virtual robot representations (DTs). Since Unity already contains a toolkit for physics simulation, we decided for simulating the Lego NXT robot with Unity physics. Simulating a robot in Unity or the gazebo simulator[5], both yield less time in testing on the real robot and especially also fewer real robots that can take damage. The Lego NXT robot simulation happens in the *Unity NXT Simulation* component. Apart from simulating the Lego NXT Robot, our framework also allows executing the written programs on real robots. The first approach to execute the program on a robot is the *Program Interpreter*. This component listens on a Bluetooth port on the robot for a program. The *Real NXTCode Translator* component can send the program in our app to the robot. This component converts the in-app program to a short format. In this way, the program can be transferred quickly. Additionally to execution via interpretation, our framework supports the Lego NXT robot execution via code generation. For this purpose, our framework has two different components. The first component is the *Lejos Code Generator*. This component generates Lejos[6] code from the internal code representation. We facilitate T4-Templates[7] for this purpose. The generated code in text format can be submitted to our second component via HTTP. The *Lejos Compilation Server* computes the HTTP request and generates a binary file.

[2]https://www.lego.com/de-de/themes/mindstorms
[3]https://www.dobot.cc/dobot-magician/product-overview.html
[4]https://www.blender.org
[5]https://gazebosim.org
[6]https://lejos.org
[7]https://github.com/mono/t4

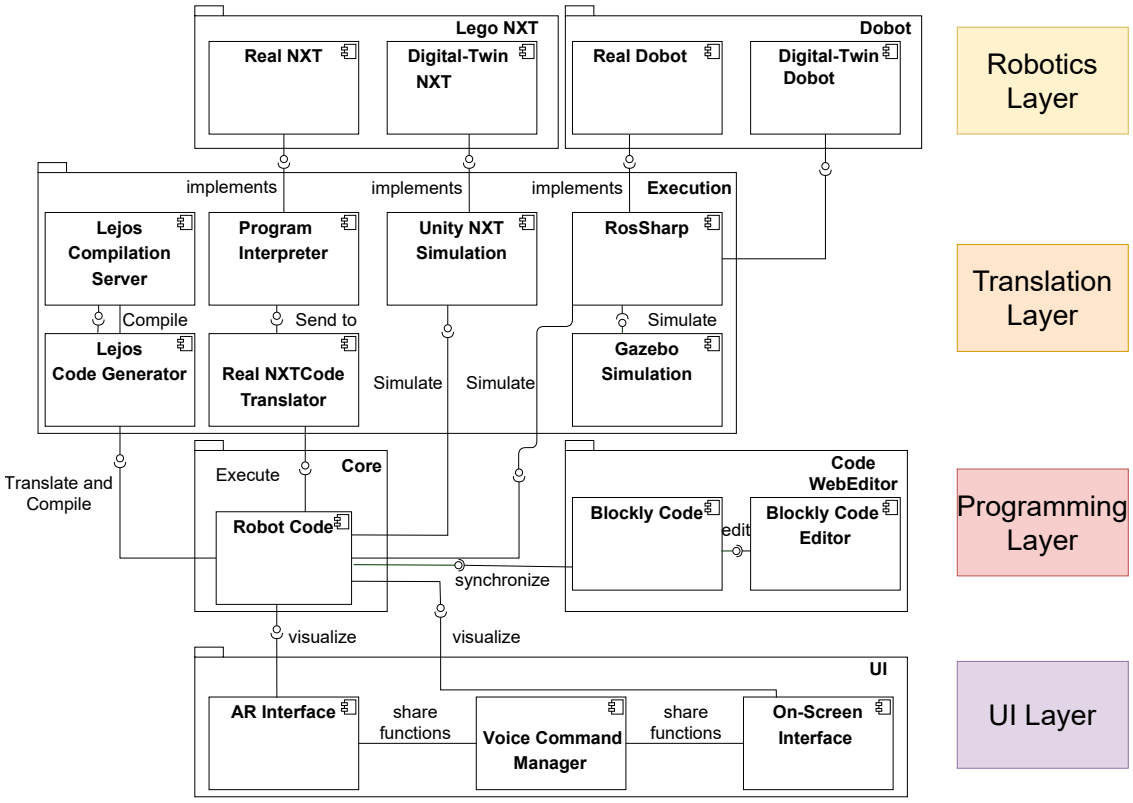

Figure 1. Architectural Overview of DART Framework

This binary file can be directly deployed on the robot. To parallelize build jobs, the *Lejos Compilation Server* builds upon Kubernetes[8].

On the *Programming Layer*, the internal structure of a robot program is defined as well as its specification. For this purpose, the *Programming Layer* consists of two main components. The *Core* component is responsible for specifying and characterizing the source of the *Robot Code* based on *RosSharp*[9]. On the other hand, we provide a component *Code Web-Editor* that consists of the subcomponents *Blockly Code* and *Blockly Code Editor* that support a traditional web-based robot programming interface based on Blockly as programming syntax.

The last layer in our architectural overview is the *UI Layer*. This layer consists of the components *AR Interface*, *Voice Command Manager*, and *On-Screen Interface*. The *AR Interface* supports the interactions for robot programming on the HoloLens 2[10] and a mobile AR device. Figure 2 (left) shows an overview of the AR interface while programming the dobot arm while Figure 2 (right) depicts a programming scenario with the mobile Lego NXT robot. As can be seen in the screenshots, the AR interface provides a block-based syntax to create and edit robot programs. The users are able to create and edit their program commands through natural interactions based on gestures. This means that the users can grab the pieces of the Blockly puzzles to create and manipulate their robot programs. In addition to that, they can also program the robot through direct interaction with the digital twin of the robot system. In the case of the dobot arm, they can program and control the robot by moving the orange sphere (see Figure 2 (left). For the mobile Lego NXT robot, this is supported through setting waypoints and drawings which tell the virtual and real robot how to move in the task environment. Besides the *AR Interface*, we support an *On-Screen Interface* to ease the handling of robot programming on mobile AR devices. Finally, the *Voice Command Manager* is integrated into the *UI Layer* to also support voice commands for robot programming. We added this component to enable a multi-modal robot programming environment.

Based on Figure 2 (right) and the example of the Lego NXT robot, in the following we describe the programming workflow and the usage of the implemented AR interface in more detail. At the start,

[8]https://kubernetes.io
[9]https://github.com/siemens/ros-sharp
[10]https://www.microsoft.com/hololens

our AR application automatically connects to the robot via Bluetooth. Afterward, the application recognizes the world coordinate space using an image marker and places the *Program Bar* (Fig. 2 (right), ①), containing the main functions, in the AR space. The buttons on the left run the virtual and real robot. On the virtual robot, programs are simulated (Fig. 2 (right), ⑤) in AR in place, showing collisions and problems without risking damage. When satisfied with the result, the user can run the program on the real robot (Fig. 2 (right), ⑥). The buttons on the right reset the virtual and real robot to the starting position. The middle of the *Program Bar* contains the programming functionalities. The first three buttons generate *Command* blocks whose style and handling are orientated on Google's Blockly, a modular, block-based programming editor. *Move Commands* (Fig. 2 (right), ②) have a command type and goal position coordinates (*0/0*). They also have a button for changing the goal (blue button), which spawns a *Selection Sphere* (Fig. 2 (right), ③) on the current coordinates. It can be moved to a new position in AR, which will automatically be translated into coordinates for the *Move Command*. Also, there are *ClawCommands* for moving the robot claw, with a goal claw position (up/down) instead of coordinates. To create a program, users can arrange the *Code Block*(s) below the *Program Bar*. They can be reordered or removed, and the route given by the current program is visualized by a line on the floor (Fig. 2 (right), ④) in real-time. To ease the creation of paths even more, there is a *Drawing* function (Fig. 2 (right), ①, 4th middle button). When used, a *Selection Sphere* appears as in the selection mode, but now, users can move it to create the complete robot path. On completion, our application simplifies the path to a manageable number of commands and adds them to the program. Progress or intermediate results can be saved for later recovery.

4. Evaluation

To evaluate the usability of the DART framework, we have conducted a user study where we analyzed the efficiency, effectiveness, user satisfaction, and cognitive workload of the AR programming interface of DART against a traditional web-based programming interface without digital twin and AR.

4.1. Participants and Setup

We invited students from a programming course via email and were able to recruit 13 students. The age range was from 21 to 30 years old. In addition, each participant was a computer science student in the bachelor's program. We asked the participants to rate their experience related to technology affinity, AR, and programming from 1 to 5, where 1 is poor and 5 is good. Technology affinity was comparatively high with an average of 4.4, which could be due to the form of education. AR experience is not as high, but with an average of 3.4. The programming experience has an average rating of 4.2, which again may be explained by the education of the various participants.

The given task for the participants was to program a Lego NXT robot in such a way that the robot starts at a starting point and drives to a target point in an "L" shaped form. While driving the "L" shaped path, when turning the curve, an obstacle should be avoided. The participants were divided into a group of seven users who used the web-based robot programming interface and a group of six users who used the AR-based programming interface.

To track how efficient the users are, we stop the time they need to fulfill a defined task. Comparing these times, we can make a statement regarding the efficiency of the two approaches. To measure the effectiveness of the software we track the errors that might occur by observation and making notes in the user study protocol. To classify the different error types, we differentiate between "Interaction" concerning and "Programming Language" concerning error types. "Interaction" concerning errors can be divided up into accidental or false interactions (Manakhov and Ivanov, 2016). An interaction error is called accidental if the cause of the error was carelessness rather than a misunderstanding, but the intent was correct. An example would be entering a greater distance for the move command than intended (11 instead of 1). False interaction errors are errors that occur because the user misunderstood a part of the interface. This means that the result does not correspond to the desired goal, although the actions he performed are correct. For example, if the user thinks that the "Start Simulation" button will also trigger the real robot, his program may execute perfectly, but the real robot will never move. "Programming Language" concerning errors can be divided up into either syntactic or semantic errors, as presented by Hristova et al., 2003 for the Java programming language, but which can also be adapted to other programming languages as the one that is used here. A syntactic error is every error that will stop the program on the real robot from executing or prevent the simulation from executing, based on a language concerning error. One example could be to use a Boolean as input for the move command. A semantic error is every logical error that prevents the robot from reaching its goal, for example turning in the wrong direction or hitting a wall. We further split the semantic errors into errors that occur with the simulation

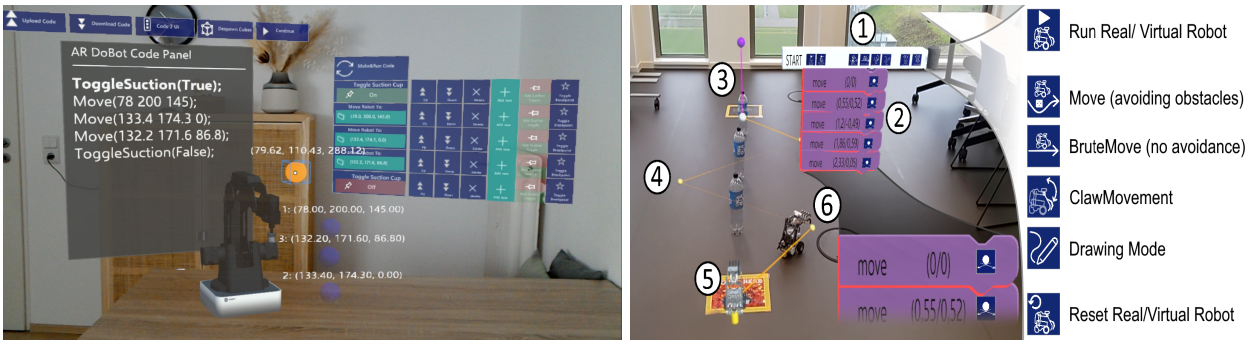

Figure 2. AR programming interface: programming the dobot arm (left) and the Lego NXT robot (right)

and errors which happen while using the real robot, as the simulation is intended to help the user try out their programs and check it for possible errors. The usability will be measured using the system usability scale (SUS) (Brooke et al., 1996) and the raw NASA task load index (TLX) (S. G. Hart and Staveland, 1988), which are a standard procedure to evaluate the user satisfaction and the mental workload of an interactive system, respectively. In addition, we ask open questions to get possible positive and negative feedback to draw conclusions.

4.2. Results

In the following, we will present and discuss the main evaluation results concerning efficiency, effectiveness, user satisfaction, and cognitive workload.

4.2.1. Efficiency As shown in Figure 3, collected data regarding efficiency shows that the average time for programming the task based on the traditional web programming interface took only 3:50 minutes, while AR took 5:31 minutes on average. One possible reason for this result is that AR users have the ability to simulate the program before actually running it, increasing the time required, but this does not fully explain the difference, as the simulation did never last longer than 20 seconds. Another reason may be that the interaction with the AR environment is not as intuitive as interacting with the web interface, since the computer is an ordinary interface. Most people know how to interact with computers and have been using similar interfaces and input mechanisms for years. In contrast, AR is a rather new approach, although the participants have had some experience with it, in total they had less interaction with it than with a classical computer interface. Another factor is that the traditional web programming interface provides the ability to copy and paste blocks of code, which reduces the time required, as the looping solutions always consisted of two very similar loops. On the other hand, AR does not support this feature, which meant that participants had to build the loops themselves. The combination of these three reasons, which are not exhaustive, can explain the difference in times.

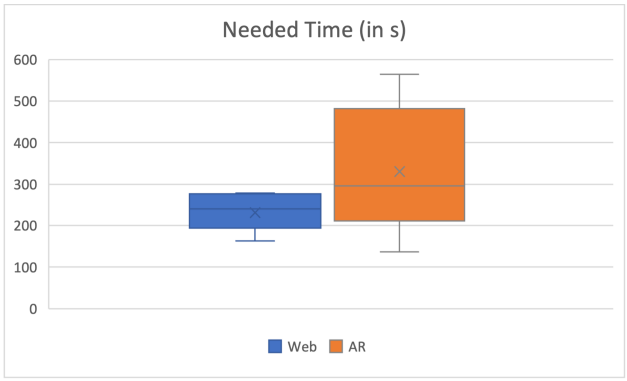

Figure 3. The efficiency of AR-based and Web-based robot programming interface

4.2.2. Effectiveness In terms of effectiveness, by looking at the average number of errors made per each run of task completion, we see that the AR users made an average of 1.2 errors. In contrast, the web group made an average of only 0.7 errors. However, when we look at the different types of errors, we see a different picture, as shown in Figure 4 (left). Syntactic errors were made on both devices: 0.7 average errors in AR versus 0.4 average errors on the web interface. Regardless of which device was used, the most common error was the incorrect use of conditions for loops, as most errors were made by indirectly evaluating Boolean values, such as ”if true,” even though this is not supported by any of the software. One of the reasons for this could be that this is a common paradigm in many programming languages such as Java or Python. Due to the high level of programming experience in the user group, this pattern

Figure 4. Effectiveness (avg. number of errors per task) of AR-based and Web-based interface

is considered standard in programming languages and therefore the participants instinctively tried to adapt this behavior. When looking at the semantic errors, a similar picture emerges: the web user made an average of 0.3 errors, while the user of the AR environment made an average of 0.5 errors. However, looking at where these errors were made, with the simulation or the robot, a difference can be seen. In the simulation in the AR environment, the users made an average of 0.5 errors, while no error occurred with the real robot. On the other hand, the web user made 0.3 errors with the real robot, as they were not able to simulate the built program. The number of errors made with the simulation may be a throw higher, but since the user had the opportunity to try out what he was doing, there were no errors on the real robot. Since the real robot should not be damaged in a real scenario, as robots can become very expensive in an industrial setting, this shows that the simulation accomplishes the desired goal by preventing the user from making fatal mistakes. In industrial production steps, these mistakes can lead to damage to the robot during the execution of the program. While executing no user of the AR environment drove the robot into one of the obstacles or make it leave the track, as these mistakes could be prevented using the simulation. In contrast, such errors were the common semantic error occuring while usage of the web-based programming interface. The second classification of errors was to divide them into accidental mistakes and errors caused by false interaction. Figure 4 (right) shows the different results. The errors made due to false interaction, i.e. that the path to the correct goal is wrong, were higher in the web interface of the software. One reason for this result could be that participants lack a view of the route and the task to be solved. Without seeing the route, it happened that participants produced semantic errors, such as driving too far, having the correct destination, but missing it due to a miscalculation of the actual distance needed. In contrast, using the AR environment the user could always see the track and task and by that prevent such false interaction mistakes. The errors made in both environments were often semantic errors, as users misinterpreted the use of conditions, as described in the discussion of syntactic errors. The other statistic is the difference between the accidental errors, which is clearly in favor of the web intereface. As mentioned earlier, this may be due to the fact that users have more experience using computers than AR interfaces. Another reason could be that entering data on the computer is easier. When entering a number, for example, the user can simply use the keyboard; in contrast, in AR, the user must use a touch Numpad, which is not as common. In general, AR users made more mistakes, but they were not as fatal as users of the web interface, leading to a safer execution of the program, concerning the robot and its surroundings.

4.2.3. User Satisfaction The aggregate results of the SUS can be seen in Figure 5. The AR-based programming interface of DART received an average aggregate score of 89.17, while the web robot programming interface received an average score of 77.5. While this already shows a difference between the two approaches, it is worth noting that the web interface results cover a wider range of scores than the AR. The range of results for the web interface was 47.5 points, while the AR had a range of only 10 points. These data show that the AR-based programming interface based on a digital twin achieved a way better user satisfaction result compared to the traditional web-based programming interface without a digital twin and AR.

4.2.4. Cognitive Workload NASA's TLX framework was used to measure the mental and physical workload of the users, and the aggregated results can be seen in Figure 6 (left). The average score of the AR user was 19, while that of the web user

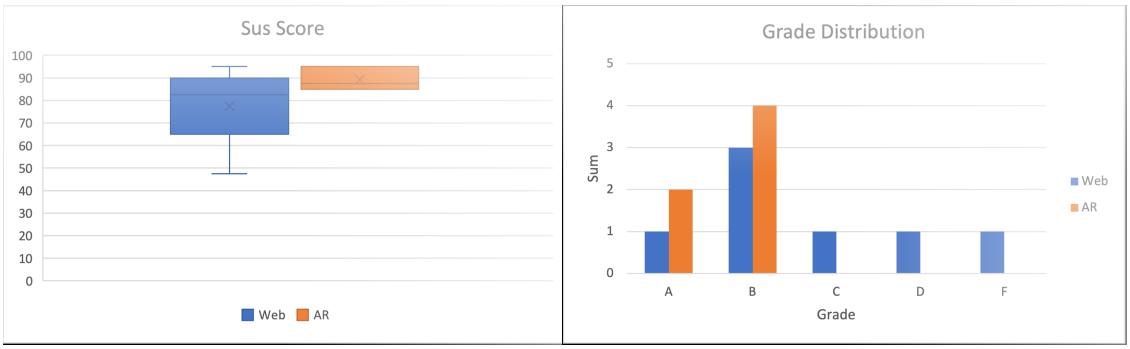

Figure 5. SUS score and grade distribution of AR-based and Web-based robot programming interface

was 22.57, which is not a large difference between the approaches. However, comparing the scores without the physical score, since the AR approach tends to have a much higher physical score due to the nature of the device used, a larger difference can be seen, with an AR average of still 19 and a Web average of 25, as shown in Figure 6 (right). This comparison clearly shows that the mental workload is lower when using AR than when using the web interface. One possible reason for this is that the user of the web interface does not see the environment in front of him, but only the screen he uses to program the robot. When the user sees the environment in front of them, they can better visualize what they need to do to successfully complete the task. Another part is the digital twin, which can be used to simulate the built program. With it, the user can try different ideas without fear of failing with the real robot. This reduces the pressure on the participant and thus reduces the mental strain required to complete the task.

4.3. Discussion

In summary, we were not able to show an advantage in the efficiency of the AR-based programming interface compared to the web interface due to several factors, such as the interaction model on the HoloLens that users were not used to. Since users made more errors with the AR programming environment, we cannot conclude that its effectiveness is better than the web interface. However, as described, the nature of the errors was different because the potential damage to the real robot can be prevented by the simulation through the digital twin in the AR environment. In addition, user satisfaction is higher in the AR environment. Last but not least, the mental workload is lower in the AR environment. We can conclude that the AR environment helps the user to solve a given task as it enhances the understanding of the environment and helps to imagine the steps that need to be done for programming the robot.

In addition to the above-mentioned points, it should be noted about this evaluation that the group of participants all have a programming background, which means that the results cannot be generalized to all possible users. Especially for inexperienced users regarding programming skills, an evaluation with layperson is necessary, as they may not understand the presented concepts as fast as users with a professional background. Besides the background of the participants, also the age and amount of the participants could lead to a bias in the results, why the above-mentioned evaluation results should be seen as observed tendencies. Thus, further user studies with larger groups of heterogeneous users are required to derive significant results about the usability of the programming environments.

5. Summary and Future Work

In this paper, we have presented a solution approach on how to combine the concept of digital twin (DT) with augmented reality (AR) to enhance the complex, cumbersome and error-prone task of robot programming. For this purpose, we have introduced the DART framework which supports an integrated DT- and AR-based robot programming environment. DART supports users to program a robot through interactive gestures, offers AR in-place program simulation through DTs of real robots, and direct building of finished programs to the real robots. We evaluated our AR-based programming approach regarding usability compared to a web-based robot programming approach without the integration of DT and AR. The evaluation showed that our approach is more usable than the conventional method, especially in terms of user satisfaction and cognitive workload, and has the potential to enrich and ease current robot programming processes.

In future work, we plan to conduct larger user studies

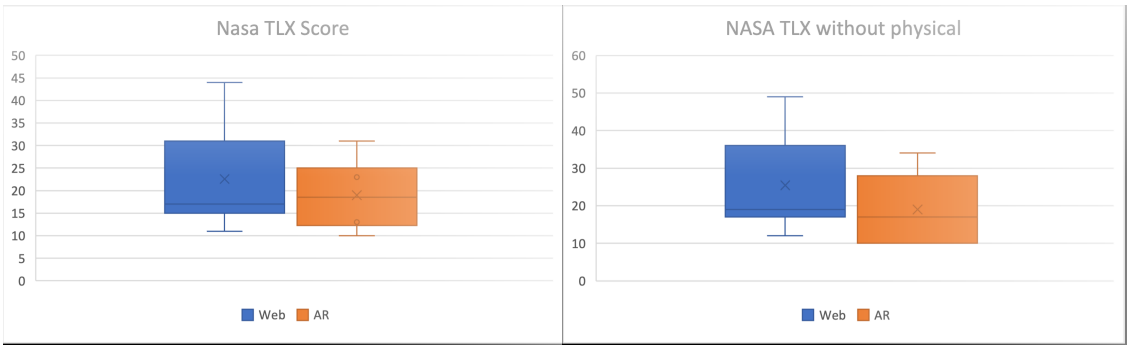

Figure 6. NASA TLX score of AR-based and Web-based robot programming interface

with heterogeneous users also covering other target platforms. Furthermore, we plan to apply our solution to further domains and robot types. Another possible extension is the inclusion of robot collaboration in our solution approach. Apart from that, the advantages of the AR interface could be used to ease the process of debugging robot code as the digital twin and AR simulation can help to find errors before executing the final code on the real robot system.

6. Acknowledgement

We would like to thank André Ortmann for his support during the implementation and evaluation of the presented approach.

References

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

Burghardt, A., Szybicki, D., Gierlak, P., Kurc, K., Pietruś, P., & Cygan, R. (2020). Programming of industrial robots using virtual reality and digital twins. *Applied Sciences*, *10*, 486.

Gadre, S. Y., Rosen, E., Chien, G., Phillips, E., Tellex, S., & Konidaris, G. D. (2019). End-user robot programming using mixed reality. *Int. Conf. on Robotics and Automation, ICRA 2019*, 2707–2713. https://doi.org/10.1109/ICRA.2019.8793988

Garg, G., Kuts, V., & Anbarjafari, G. (2021). Digital twin for fanuc robots: Industrial robot programming and simulation using virtual reality. *Sustainability*, *13*(18), 10336.

Hart, J. W., DePalma, N., Pryor, M. W., Hayes, B., Kruusamäe, K., Mirsky, R., & Xiao, X. (2021). Exploring applications for autonomous nonverbal human-robot interaction. *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, 728–729.

Hart, S. G., & Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology* (pp. 139–183). Elsevier.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting java programming errors for introductory computer science students. In S. Grissom, D. Knox, D. T. Joyce, & W. P. Dann (Eds.), *Proc. of the 34th SIGCSE technical symposium on computer science education* (pp. 153–156). ACM. https://doi.org/10.1145/611892.611956

Josifovska, K., Yigitbas, E., & Engels, G. (2019). A digital twin-based multi-modal UI adaptation framework for assistance systems in industry 4.0. In M. Kurosu (Ed.), *Human-computer interaction HCI* (pp. 398–409). Springer.

Lipton, J. I., Fay, A. J., & Rus, D. (2018). Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing. *IEEE Robotics and Automation Letters*, *3*(1), 179–186. https://doi.org/10.1109/LRA.2017.2737046

Manakhov, P., & Ivanov, V. D. (2016). Defining usability problems. In J. Kaye, A. Druin, C. Lampe, D. Morris, & J. P. Hourcade (Eds.), *Proc. of the 2016 CHI conference on human factors in computing systems* (pp. 3144–3151). ACM. https://doi.org/10.1145/2851581.2892387

Muszyńska, M., Szybicki, D., Gierlak, P., Kurc, K., Burghardt, A., & Uliasz, M. (2019). Application of virtual reality in the training of operators and servicing of robotic stations. *Working Conference on Virtual Enterprises*, 594–603.

Olabi, A., Damak, M., Bearee, R., Gibaru, O., & Leleu, S. (2012). Improving the accuracy of industrial robots by offline compensation

of joints errors. *2012 IEEE international conference on industrial technology*, 492–497.

Rosen, E., Whitney, D., Phillips, E., Chien, G., Tompkin, J., Konidaris, G. D., & Tellex, S. (2019). Communicating and controlling robot arm motion intent through mixed-reality head-mounted displays. *Int. J. Robotics Res.*, *38*(12-13). https : / / doi . org / 10 . 1177 / 0278364919842925

Shepherd, D. C., Francis, P., Weintrop, D., Franklin, D., Li, B., & Afzal, A. (2018). [engineering paper] an IDE for easy programming of simple robotics tasks. *18th IEEE Int. Work. Conf. on Source Code Analysis and Manipulation*, 209–214. https : / / doi . org / 10 . 1109 / SCAM . 2018.00032

Shepherd, D. C., Kraft, N. A., & Francis, P. (2019). Visualizing the "hidden" variables in robot programs. *Proceedings of the 2nd International Workshop on Robotics Software Engineering, RoSE 2019, Montreal, QC, Canada, May 27, 2019*, 13–16. https://doi.org/10.1109/RoSE.2019.00007

Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., & Sui, F. (2018). Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, *94*(9), 3563–3576.

Thanigaivel, N. K., Ong, S. K., & Nee, A. (2019). Augmented reality-assisted robot programming system for industrial applications. *Robotics and Computer-Integrated Manufacturing*, *61*. https://doi.org/10.1016/j.rcim.2019.101820

Theofanidis, M., Sayed, S. I., Lioulemes, A., & Makedon, F. (2017). Varm: Using virtual reality to program robotic manipulators. *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, 215–221.

Weintrop, D., Shepherd, D. C., Francis, P., & Franklin, D. (2017). Blockly goes to work: Block-based programming for industrial robots. *2017 IEEE Blocks and Beyond Workshop (B B)*, 29–36. https : / / doi . org / 10 . 1109 / BLOCKS . 2017 . 8120406

Weintrop, D., Afzal, A., Salac, J., Francis, P., Li, B., Shepherd, D. C., & Franklin, D. (2018). Evaluating coblox: A comparative study of robotics programming environments for adult novices. In R. L. Mandryk, M. Hancock, M. Perry, & A. L. Cox (Eds.), *Proc. of the 2018 CHI conference on human factors in computing systems* (p. 366). ACM. https : / / doi . org / 10 . 1145/3173574.3173940

Whitney, D., Rosen, E., Ullman, D., Phillips, E., & Tellex, S. (2018). Ros reality: A virtual reality framework using consumer-grade hardware for ros-enabled robots. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–9. https://doi.org/10.1109/IROS.2018.8593513

Zhang, C., Zhou, G., He, J., Li, Z., & Cheng, W. (2019). A data-and knowledge-driven framework for digital twin manufacturing cell. *Procedia CIRP*, *83*, 345–350.