



## RESEARCH ARTICLE

# Training industrial end-user programmers with interactive tutorials

Nico Ritschel<sup>1</sup>  | Anand Ashok Sawant<sup>2</sup>  | David Weintrop<sup>3</sup> | Reid Holmes<sup>1</sup> | Alberto Bacchelli<sup>4</sup> | Ronald Garcia<sup>1</sup> | Chandrika K R<sup>5</sup> | Avijit Mandal<sup>5</sup> | Patrick Francis<sup>6</sup> | David C. Shepherd<sup>7</sup>

<sup>1</sup>Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada

<sup>2</sup>Department of Computer Science, University of California, Davis, Davis, California, USA

<sup>3</sup>College of Education, University of Maryland, College Park, Maryland, USA

<sup>4</sup>Department of Informatics, University of Zurich, Zurich, Switzerland

<sup>5</sup>ABB Corporate Research, Bangalore, Karnataka, India

<sup>6</sup>UserVoice Inc., Raleigh, North Carolina, USA

<sup>7</sup>Department of Computer Science, Virginia Commonwealth University, Richmond, Virginia, USA

## Correspondence

Nico Ritschel, Department of Computer Science, University of British Columbia, Vancouver, BC V6T1Z4, Canada.  
Email: [ritschel@cs.ubc.ca](mailto:ritschel@cs.ubc.ca)

## Funding information

ABB Corporate Research; National Science Foundation, Grant/Award Number: 2024561; Natural Sciences and Engineering Research Council of Canada

## Abstract

Newly released robot programming tools have made it feasible for end-users to program industrial robots by combining block-based languages and lead-through programming. To use these systems effectively, end-users, who usually have limited or no programming experience, require training. To train users, tutoring systems are often used for block-based programming—some even for lead-through programming—but no tutorial system combines these two types of programming. We present CoBlox Interactive Tutorials (CITs), a novel tutoring approach that teaches how to use both the hardware *and* software components that comprise a typical end-user robot programming environment. As users switch between the two programming styles, CITs provide them with extensive scaffolding, give users immediate feedback on missteps, and provide guidance on next steps. To evaluate CITs, we conducted a study with 79 industrial end-users using a programming environment released by ABB Robotics that compares our approach to training with training videos, the most commonly used training in industry. This study, one of the largest to date on training professional end-users, found that CIT-trained users authored more correct programs in less time than video-trained users. This shows that a tight integration of hardware and software concepts is crucial to training end-users to program industrial robots.

## KEYWORDS

end-user programming, programming support, robotics, tutorials, user study

## 1 | INTRODUCTION

Robotic automation, now more than ever, has the potential to transform not just manufacturing processes, but society as a whole. Industrial robots have become substantially cheaper and more capable, and this trend is likely to continue. As smaller organizations can increasingly afford robots, more industries can benefit from enhanced automation<sup>1</sup> and robots are being used for a wider range of tasks than ever before. As this success snowballs, the number of robots deployed in professional environments is rapidly growing.<sup>2</sup>

**Abbreviations:** CITs, CoBlox Interactive Tutorials; ITS, Intelligent Tutoring System; SUS, System Usability Scale.

Nico Ritschel and Anand Ashok Sawant contributed equally to this study.

As the demand for robots increases, so does the demand for employees that can program them. However, traditional industrial robot programming is difficult; it requires developers that are able to translate the robot's physical properties, environment, and actions into code. Experts for this kind of task are expensive, and traditional programming education cannot keep up with the steeply increasing number of industrial robots.<sup>3</sup>

If more employees could learn how to program robots, deploying new robots and reprogramming existing ones would become cheaper and easier. However, most employees lack programming experience and formal training and would struggle to use traditional programming tools designed for experts. Instead, these end-users need easy-to-learn programming environments, with easy-to-understand training that teaches them how to program practical routines quickly and effectively.

Researchers have employed a wide range of different strategies to make robot programming tools more friendly to end-users.<sup>4,5</sup> Many end-user tools are built around simplified, beginner-friendly programming languages, providing a software solution.<sup>6-8</sup> Other systems focus on the hardware, letting users define programs via lead-through programming, where the user physically guides the robot through the steps of the task.<sup>9-11</sup> Recent research has shown that an environment that combines both of these approaches can overcome many of their individual limitations.<sup>12</sup> The presented system, called *CoBlox*, integrates a block-based programming language with lead-through programming, and has been released by ABB Robotics as Wizard Easy Programming.\* An initial evaluation of *CoBlox* has found it to be substantially easier to learn and use for end-users than other state-of-the-art robotics tools.<sup>12</sup> However, this evaluation was performed on a simulator only, trivializing the substantial lead-through aspects of the programming tasks. To program a real robot, users must learn to work with both the software and hardware interfaces of the programming environment, and combine how both components work in tandem.

Most work on training programmers focuses on school-aged learners (ages 5–22) instead of adult learners, which can have different needs and motivations. For example, professional learners are more likely to be driven by specific, immediate work needs than a general interest in acquiring new knowledge and skills.<sup>13</sup> Professionals are also more likely to use trial-and-error approaches to learn new tools and technologies.<sup>13</sup> Learner-centric approaches and short, targeted training sessions with realistic tasks therefore seem most likely to succeed,<sup>14,15</sup> but few studies have evaluated such training methods in industrial settings.

This article introduces a tutorial system for robot programming that provides hands-on training and bridges the gap between hardware and software programming interfaces. The system, called *CoBlox Interactive Tutorials (CITs)*, is embedded into *CoBlox* and assists users with authoring software programs and manipulating the robot's hardware. CITs support active learning, which previous research has shown to be effective for training end-users for their daily tasks.<sup>16,17</sup> The system further embeds instructions and feedback directly into the programming environment, similar to other state-of-the-art training tools.<sup>18-21</sup> However, unlike existing tutoring systems that teach either traditional programming or hardware interactions,<sup>22,23</sup> CITs target both modalities at once and shows users how they can transition between them. In the context of robot programming, CITs therefore provide a richer, more integrated learning experience than existing tutoring systems.

We have investigated whether the integrated approach of CITs can benefit end-users by evaluating them in a comparative study. This study compared the performance of 79 professional adult end-users on realistic tasks after they were trained using either CITs or a conventional training video. The study's results show that most users can quickly learn how to use the traditional programming aspects of *CoBlox*, independent of the training method. However, some of the users trained via video struggled with the physical aspect of robot programming, causing them to solve tasks slower, or not at all. Participants that used CITs were significantly faster and more successful overall, and all of them made at least some progress on every given task. Our results therefore demonstrate that CITs are a valuable extension to the *CoBlox* system and are particularly helpful for users that would struggle otherwise, allowing them to successfully master the hardware *and* software aspects of robot programming. This study further reinforces the findings of a previous, smaller study<sup>12</sup> that found *CoBlox* to be easy to learn and use overall.

This work advances our understanding of how to design instructional support for programming environments with both hardware and software components. It further demonstrates how tools and training materials can effectively target adults that have little to no prior programming experience. Finally, it contributes one of the largest studies to date on training methods for professional end-users.

\*<https://new.abb.com/products/robotics/application-software/wizard>

## 2 | RELATED WORK

This work integrates a novel tutorial system into the CoBloX programming environment, which we present in detail in Section 3.1. The original CoBloX publication outlined previous research on making robot programming and general programming more accessible for end-users.<sup>12</sup> This work focuses on supporting industrial end-users through automated and integrated tutoring. In this section, we therefore summarize and categorize this strand of research and outline how our work contributes to its study.

### 2.1 | Tutoring systems and tutorials: An overview

A large body of human-computer interaction work has explored ways to help people understand and use novel technologies (e.g., References 24 and 25). The effectiveness of human tutors in education has long been known,<sup>26</sup> and substantial effort has been invested in matching this impact using automated technology.<sup>27,28</sup> Intelligent Tutoring Systems (ITSs) aim to aid or replace human tutoring by automatically generating a level of rich, customized feedback similar to that of a human tutor.<sup>29</sup> While tutoring systems can teach a wide range of tasks across many domains, we focus here on ITSs that teach programming or programming-related tasks.

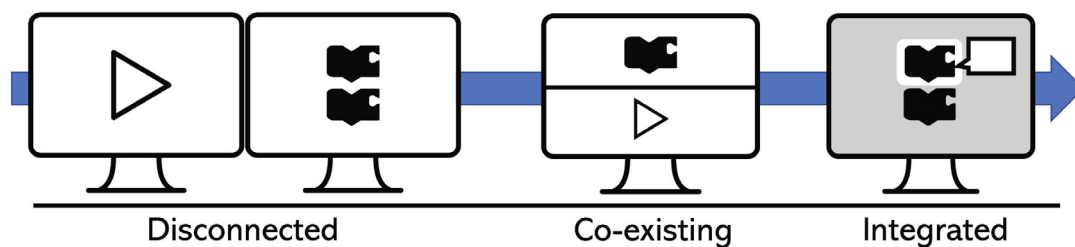
Previous work has extracted design guidelines for programming tutoring systems from the large body of both theoretical and applied work in this area.<sup>30</sup> Based on their observations, the authors generated a list of recommendations for tutoring systems. One example recommendation is to engage learners in active learning by making them write code themselves. Another recommendation is to guide them with short, goal-directed tasks.

As the design of tutoring systems has evolved over time, so has their level of integration into the systems they target. Figure 1 illustrates the three most commonly found levels of ITS integration from left to right: *Disconnected* tutoring systems are entirely detached from the system they are training learners to use. *Coexisting* systems feature some superficial connection with the target system but are still operating mostly independently. *Integrated* systems are deeply embedded into the system they are targeting and can access and modify both its internal state and presentation.

In the following subsections, we illustrate these three levels of integration further. As disconnected and coexisting tutoring systems are ubiquitous, we focus on a few representative examples. For integrated systems, which are most closely related to our work, we provide a more exhaustive overview of relevant previous research. Only a small body previous work has focused on tutoring for physical programming as provided by CITs. However, we conclude by describing these previous attempts and how they differ from the work we present here.

### 2.2 | Disconnected and coexisting tutoring systems

Disconnected tutoring systems, such as Google's CSFirst<sup>31</sup> curriculum, have the advantage that they can be developed and maintained independently of the target system. They ask users to watch or read training materials in a separate application that has no relation to the programming environment. Notably, this form of tutorial can utilize a wide range of media content, such as videos or screenshots of the programming environment they target. For example, the official tutorial for LabVIEW<sup>32</sup> makes extensive use of such materials to refer to specific interface elements or environment features.



**FIGURE 1** Examples of training, ordered by level of integration into the programming environment: Disconnected training (left) separates instructions and programming; in coexisting training (middle), the two components are loosely connected; only in an integrated environment (right), training and programming environment can interact to give feedback and guide users.

However, due to the separation between the teaching and application environment, this type of system has significant usability drawbacks.

Disconnected tutoring systems require users to switch between two environments that provide entirely different learning contexts. This switch of contexts also affects users on a mental level, and causes strain as it requires them to actively bridge between the instructions and the actual programming system.<sup>21</sup> Furthermore, disconnected systems cannot provide users with any feedback about their progress since they do not have access to any information about the programming environment. Therefore, the task of judging their work and tracking their progress is also made the responsibility of the learner. Such an expectation can overwhelm users and slow down their learning progress.<sup>33</sup>

Popular examples of coexisting tutoring systems are Code.org's Hour of Code program<sup>34</sup> and Scratch Tutorials.<sup>35</sup> These systems embed instructions directly into their respective code editor as pop-ups. However, despite this embedding, systems in this category do not interact with the programming environment beyond the superficial user interface integration. In particular, coexisting systems do not react to changes in the programming environment or give feedback on the user's programs.

Due to the predetermined nature of coexisting tutoring systems and the inability to react to real parameters of the programming environment, tutorial designers have to choose example tasks that can be completed in a highly deterministic fashion. In the context of robot programming, this means that the robot's physical set-up needs to be exactly predetermined or instructions need to be written in a way that captures all possible situations. However, even in these cases, if a user makes a mistake, this can lead to situations where they do not have a clear path to the intended outcome of the tutorial anymore. Coexisting systems further cannot directly highlight or interact with user interface elements. Instead, coexisting tutorial systems typically use screenshots or figures as a proxy. This can confuse users if they mistake images for real user interface elements,<sup>33</sup> an effect that is amplified by the visual integration of tutoring system and programming application.

In summary, while disconnected and coexisting tutoring systems are often more feasible and easier to create, they come with design limitations that can substantially impact the learning of their users. However, despite these limitations, non-integrated tutoring can have a place in certain learning settings, as we describe in Section 6.2.

## 2.3 | Integrated tutoring systems

Compared to disconnected and coexisting tutoring systems, integrated systems provide a substantially deeper integration between tutorial and development environment. To our knowledge, the block-based programming system Alice<sup>36</sup> was the first to provide a substantial integration. Alice's approach, called stencils, dims and deactivates most of the user interface, only showing the relevant component(s) for the next step in a given tutorial. As users follow each step of the tutorial, instructions move on automatically and always focus on the next step necessary to complete a task. Alice's Stencil approach was highly influential and inspired subsequent work.<sup>18,19,37,38</sup>

Stencils illustrate the advantages of integrated tutorial systems: they can guard irrelevant components and therefore prevent users from many missteps that they could make. They can also draw a user's attention to certain interface elements, and reduce the perceived amount of visual clutter on the screen. The entirely guided teaching approach comes at a cost: studies found that users learning via this method were less confident that they could write their own programs since they had little autonomy over their actions.<sup>20</sup> Furthermore, because users make less mistakes they miss the valuable learning that mistakes and their correction can provide.<sup>39</sup>

A more modern wave of integrated systems, like RoboBlockly<sup>40</sup> or GamiCAD,<sup>41</sup> have modified Alice's stencil approach to be less invasive. They highlight or point at interface elements, but also allow users to ignore the instructions or make adjustments. This means that users can make mistakes, which the systems address by providing immediate feedback and a suggestion on how to fix their program. These design elements aim to give users more confidence when they work on real tasks after completing the tutorial, but empirical evidence on their benefits is limited.

This work builds on the most recent techniques for integrated tutorials. For the virtual programming steps, such as selecting and assembling command blocks, CITs highlight user interface elements, but do not block off the remaining interface. However, unlike other systems, CITs extend this paradigm further, adding support for the physical programming aspects of CoBlox. In addition, we have evaluated CITs on a substantially larger number of industrial end-users than previous work.

## 2.4 | Tutoring systems for physical programming

While there exist many tutoring systems for traditional on-screen programming, a much smaller body of work targets physical programming of hardware as it is used in the CoBlox programming environment. These tools typically provide instructions in the form of photos, videos or animations of the physical actions they expect the user to perform next. However, they typically lack any integration with the physical hardware they target, and therefore cannot provide any automated feedback. To our knowledge, all of these tools can therefore be categorized as *disconnected* or at best *coexisting* based on our previously established categorization.

The recent work on the tutoring system *CircuitStyle*<sup>42</sup> is a good example how challenging it can be to give feedback for tasks executed on hardware. In this system, users have to follow specific style guidelines as they build a circuit, which are provided by the system and illustrated with pictures. However, the system cannot verify that users have actually followed the guidelines and relies on self-assessment or feedback from a human instructor instead. We therefore consider this system as *disconnected* from the hardware it targets.

Some tutoring systems in the circuit design space, like *HeyTeddy*<sup>22</sup> or *ElectroTutor*,<sup>23</sup> have formalized this indirect feedback mechanism further. These systems explicitly ask the user to either verify their solution or enter specific physical parameters, such as voltages, to check that specific tutorial steps have been completed successfully. While relying on the user to manually collect and transfer information between the systems can enable some form of feedback, this mechanism introduces a substantial overhead that can make the system tedious to use. Therefore, such systems have to consider the trade-off between gathering information for accuracy and not interrupting the user's focus.

Previous tutoring systems for physical programming resemble those that we previously classified as *coexisting*, both in terms of the degree of interaction between the tutoring and the hardware system, and considering the resulting distraction and lack of immediate feedback to users. Nonetheless, previous evaluations have demonstrated them to be effective in supporting the learning of students. However, this evidence is limited as it is based on small experiments that usually involved students rather than industrial participants. This work proposes a tutorial system that targets both physical and virtual programming with an *integrated* approach, and it also addresses the lack of a realistic evaluation on industrial end-users.

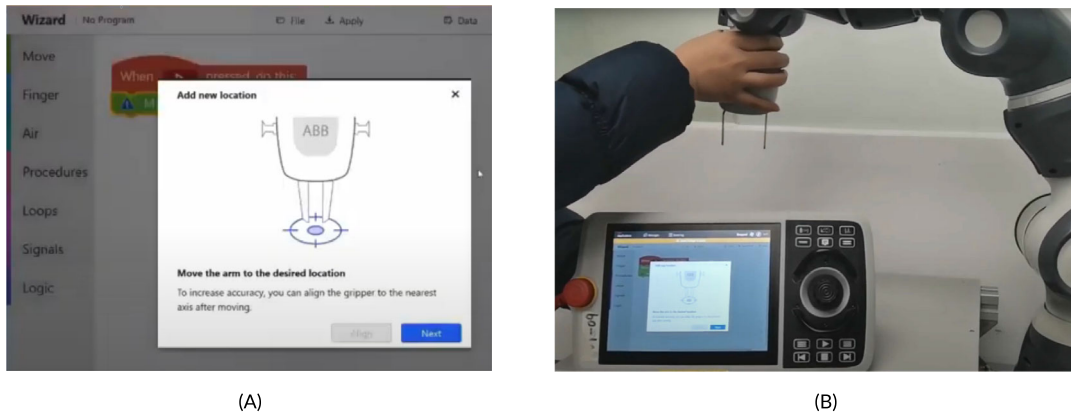
## 3 | APPROACH

As we discussed in Section 2, many existing tutorial systems are not or only loosely integrated with the programming environments they target. For physical programming environments in particular, we did not identify any existing work that matches our definition of an *integrated* tutoring system. CoBlox Integrated Tutorials address this shortcoming through a deep integration with CoBlox, a programming environment that combines traditional block-based programming with physical lead-through programming. This integration aims to help adult novices learn both aspects of CoBlox more effectively than if they used a traditional, less integrated tutoring system. This Section introduces CoBlox in more detail to provide background on its existing features, and then explains the novel aspects of CITs and the motivating factors behind their design.

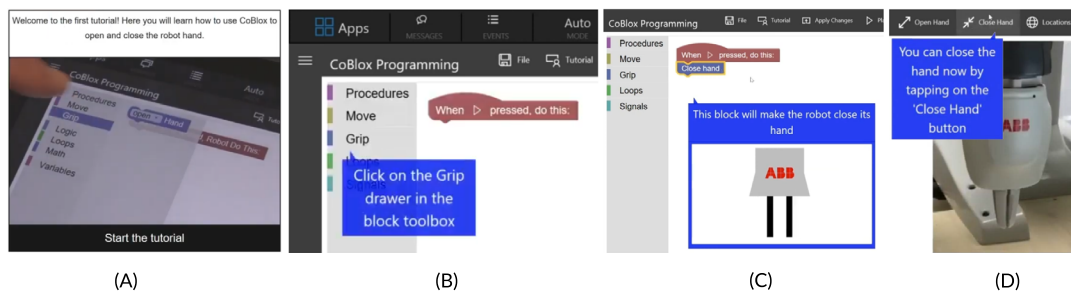
### 3.1 | A brief tour of CoBlox

The CoBlox environment<sup>12</sup> allows end-users to program one-armed industrial robots through a block-based programming environment as well as by manipulating them physically via lead-through programming. CoBlox provides a domain-specific language with high-level commands such as `Move quickly to <location>` and `Open hand`. It also provides predefined templates for common routines such as pick-and-place. Users can adapt these routines to a specific task by setting the target location of each move instruction.

Unlike most expert tools, CoBlox never requires users to specify physical locations via code. Instead, users can capture these locations by manipulating the robot arm in lead-through mode, as illustrated in Figure 2B. This multi-modal programming approach allows novices to create programs faster than using other end-user programming tools. In a study, users gave CoBlox higher scores for usability, learnability, and overall satisfaction than any other evaluated tool.<sup>12</sup> Yet, despite this evidence of CoBlox' effectiveness, users of the tool still face challenges:



**FIGURE 2** Illustration of *hardware–software coordination* when defining a location. Note that these images and video are from a productized but nearly identical version of CoBlox, called Wizard Easy Programming. (A) On-screen instructions on how to define locations. (B) A user defining a location by moving the robot arm



**FIGURE 3** Notable features of the CoBlox Interactive Tutorial system as referenced by our walk-through. A video demonstrating the entire tutorial is available at <https://youtu.be/QRqtT5Jp3ZA>. (A) Introduction to the touch screen controls. (B) In-place prompts with instructions. (C) Animated on-screen illustration. (D) In-hardware demonstration

1. Users must recognize when they have to manipulate the robot hardware and switch to a special *lead-through* mode used for defining new positions.
2. While manipulating the robot, users may need to return to the programming environment to refer to previously defined locations and plan the robot's movement path.
3. While manipulating the robot, users may need to use the programming environment to open or close the robot's gripper hand, which is not possible though physical interaction.
4. Finally, users must finalize their desired location by disabling lead-through mode and saving the target location.

These challenges cannot be directly addressed by simplifying the CoBlox environment. For example, if CoBlox were to automatically enable or disable lead-through mode, then users might lose track of the current status of the robot. They might move the robot arm by accident, or damage it if they attempt to move it while lead-through is disabled. CITs therefore support users by teaching them when and how they can combine physical hardware programming techniques with traditional software programming.

### 3.2 | A brief tour of CITs

To demonstrate the design of CITs, this subsection presents a walk-through of the first tutorial. This tutorial introduces users to opening and closing the robot's hand. Figure 3 illustrates some of the individual tutorial steps.<sup>†</sup> Some features and design elements are discussed in more detail in the subsequent subsections.

<sup>†</sup>A video version of this walk-through is available at <https://youtu.be/QRqtT5Jp3ZA>

After users have launched the tutorial, the system shows users a brief animation of a person dragging a block using a touch screen (Figure 3A). This animation is intended to naturally demonstrate to new users how they can interact with the touch-screen interface.

After users start a tutorial, it presents them with an instruction prompt (Figure 3B). For the first tutorial, this prompt reads: “Click on the Grip drawer in the block toolbox.” The prompt directly points at the interface element that the user is intended to click. When the user clicks on the Grip drawer, it opens to reveal the available set of grip-related commands. As soon as the Grip drawer is open, a new tutorial prompt appears instructing users to drag the `Close hand` block onto the canvas.

Once users have dragged the block onto the canvas, the next prompt appears, telling users to attach the new block to the main block. Upon attaching `Close hand` to the main block, an animation appears that shows users an animation that demonstrates this command’s effect on the robot (Figure 3C). After the animation finishes, a new prompt points users to the `Close hand` button on the upper right side of the application. Pressing this button allows users to see the command’s effect on the robot hardware immediately (Figure 3D). It also introduces users to one of several interface elements that allow them to immediately control the robot.

As the first tutorial continues, users follow prompts to add an `Open hand` block to the previous one. Finally, the tutorial shows users how they can run their programs. After saving their progress, users are prompted to open the execution pane, where they can control how the robot executes of their program. The first tutorial only teaches users how to run their program regularly, but later ones guide them through debugging features like step-by-step execution.

### 3.3 | Key CIT features

The design of CITs emphasizes virtual and physical integration with the CoBlox programming environment. In the following we discuss how this integration benefits users through *embedded instructions* and *immediate feedback* for virtual and physical programming steps.

#### 3.3.1 | Embedded instructions

Building on the innovations of stencils,<sup>20</sup> embedded instructions direct users by directly highlighting user interface elements. For instance, in Figure 3B, the prompt “Click on the Grip drawer in the block toolbox” is attached directly to the drawer it refers to. This allows CITs to avoid proxies like images or videos that many of the tutoring systems we discussed in Section 2 use. However, unlike the Stencils approach, CITs allow users to make mistakes, which can provide valuable learning opportunities. For example, the tutorial does not stop users from accidentally selecting incorrect blocks.

For physical programming interactions, CITs deliberately switch between proxy animations, videos, and the physical demonstration of effects. For example, in Figure 3A, the tutorial shows a brief video that demonstrates how to use the touch screen to drag blocks. In Figure 3D, it asks users to try out a button and see the effect on the robot’s physical hand. In Figure 3C, on the other hand, the tutorial uses an animation. This proxy was chosen intentionally to avoid the misinterpretation that every attachment of blocks immediately triggers robot movements.

#### 3.3.2 | Immediate feedback

Instructions in CITs are not just inline but also provide immediate feedback. After every step that a user executes correctly, the next instruction is displayed automatically. For example, after the user clicks on the Grip drawer in Figure 3B, the tutorial immediately switches to the next instruction. Unlike other tutoring systems, CITs also provide direct, contextual feedback for mistakes. For example, if a user places an incorrect block, the tutorial instructs them to delete the blocks and revert the program back to its intended state. This experience of making a mistake and fixing it can increase the ability of beginners to address a similar situation when they encounter it at a later point in time.<sup>43</sup>

CITs also provide feedback for physical programming interactions. For example, if a user programs the robot to move along a path that is obstructed, the arm stops as soon as it hits an obstacle. The tutorial highlights the command that failed on the screen and presents users with two possible ways to resolve this issue: Either they can redefine the start and end

points of the movement, or they can retry the movement after removing the obstacle. For the latter option, users learn that they can gently physically tap the robot to resume an interrupted program.

Many of the previously discussed features of CITs aim to ease the mental burden for users as they coordinate the virtual and physical components of CoBlox. The most difficult transition for new users occurs, however, when they define new locations. Due to the unpredictable physical environment of the robot, the feedback that CITs provide cannot be as immediate as for other programming steps: Without running the program first, there is no way to check if a program works as intended. Even when an error is found while running the program, it is often difficult to precisely explain what went wrong. This interaction is likely to confuse or frustrate new users.

CITs support users when defining locations by tightly coordinating the virtual and physical components of CoBlox. When users click on the **New Location** button of a movement command, CITs dim the entire CoBlox interface to focus the user's attention on the instructions. As Figure 2A shows, this dialog features an animation and instructions that encourage the user to physically move the robot arm to an indicated position. As the dialog appears, the robot arm further makes an audible click, signaling to the user that manual movements are now enabled. This click is used throughout the tutorials and also occurs when users use CoBlox without CITs, giving users a consistent hint when they have to physically interact with the robot. After users have moved the robot arm as shown in Figure 2B, they can press the 'Next' button to capture the position of the robot arm. After the CoBlox environment asks them to name the new location, the tutorial continues with on-screen prompts. When locations are initially introduced, the tutorial immediately lets users test the resulting movement to keep the feedback loop as tight as possible.

## 4 | EXPERIMENTAL EVALUATION

To evaluate CITs, we conducted a comparative user study in which users programmed an industrial robot in the CoBlox environment after being trained with either CITs or a conventional training video.

### 4.1 | Participants

The target audience of CITs and the underlying CoBlox environment are adults with little or no prior robotics or general programming experience. To find participants that fit these criteria, we reached out to employees at a large office site of a multinational engineering conglomerate in Bangalore, India. We recruited participants from a diverse set of office professions (e.g., engineers, software developers, technical administrators). We intentionally included a range of professional profiles to understand how CITs support users with varying sets of prior experiences and professional backgrounds.

A total of 90 participants participated in the study, of which 79 completed all aspects of the protocol and are thus included in the presentation of our results. Of the 11 participants that did not complete the study protocol, 2 decided to withdraw from the study before the end of their training. Another 9 had to abort the study because they encountered significant technical difficulties that we were eventually able to trace down to a defective robot hardware component that had to be replaced. To avoid biasing our results, we have excluded these participants from our evaluation.

The average age of our participants was 30.6 years (SD 6.5 years), and of the 78 participants who indicated their gender, 53 (68%) were male and 22 (28%) were female. Participants came from a range of backgrounds: 35 (45%) were engineers, 15 (19%) researchers, 12 (15%) software developers, and the rest were either administrative staff or worked in product support. Only 7 (9%) of the participants had any prior experience with programming robots (average 3.7 years, SD 2.6 years).

### 4.2 | Research questions and study protocol

With our study, we set out to answer the following research questions:

- RQ1:** How long do participants take to complete CITs in comparison to video training that covers the same amount of content?
- RQ2:** Do CITs improve the performance of end-users compared to traditional video training as they solve robot programming tasks in CoBlox?



**RQ3:** Which of the different aspects and modalities required to program robots in CoBlox are particularly affected by training via CITs?

**RQ4:** Do CITs change the perception that end-users have on training and the overall usability of CoBlox?

To address these questions, we designed a user study that participants could complete in 60–90 min. After giving consent and spending about 15 min on training, participants had up to 35 min to solve a series of three tasks. These tasks were primarily designed to gain insights into RQ1 and RQ2. After they completed the tasks, we asked participants to fill out a brief usability questionnaire, an open-ended feedback form, and a demographic survey, for which we did not impose a fixed time limit. The usability questionnaire and feedback form aimed to answer RQ3. Throughout the study, participants worked on a computer next to the physical robot, so that they could interact with it during programming.

#### 4.2.1 | Training

At the beginning of our study, we randomly divided participants into two conditions. Half of our participants were trained interactively using CITs and the other half by watching a training video. We chose to use an introduction video as our baseline comparison because videos are the most popular training solution for software and hardware systems.<sup>44</sup>

We asked the participants of the tutorial condition to complete a sequence of four tutorials: (1) using the robot hand, (2) moving the arm, (3) picking and placing an item, and (4) calling a procedure. The four tutorials were designed to cover the concepts required to accomplish the tasks put forth in the second half of the experimental procedure. They were designed to take around 15 min in total to complete, but we did not impose a strict time limit on their completion.

The training video that we showed to participants of the video condition had the same structure and showed the steps covered by the four CITs in the same order. Instruction or description texts in the CITs were replaced by voice-over explanations in the video. For each programming step, the video showed both the programming environment and all necessary user interactions, including all the necessary lead-through programming steps. Participants of the video condition were given access to the programming environment during their training and were allowed to explore the system or familiarize themselves with it. They were however not given any explicit instructions or asked to replicate the tasks they had seen in the videos.

The training video was 10 min 51 s long. It is therefore shorter than the expected duration of the four tutorials. Observing early participants, we noticed that they tend to take longer to perform tutorial steps themselves than when they are demonstrated in the training video. While we allowed participants to pause or rewind the video, none of the participants made use of this opportunity. Participants did also not spend a substantial amount of time exploring the programming environment during or after their training. We therefore assume that the pace of our video was appropriate for them to follow each step and we decided against further measures to equalize training times, such as artificially slowing down the video.

#### 4.2.2 | Tasks

After participants finished their training, we asked them to complete a series of pick-and-place tasks using lego-like objects. Figure 4 demonstrates the set-up of our experiment. In a pick-and-place task, the robot arm moves an item between two locations while navigating a potentially constrained physical environment. We chose this type of task since it is one of the most common use cases for industrial robots. Pick-and-place tasks are also well-suited for a user study since they are easy to understand conceptually, but nontrivial to execute since users must switch between using the programming environment and manually positioning the robot.

We asked participants to complete a total of three tasks in a fixed order. Each of the tasks had a time limit, both for logistical reasons, and to limit the number of possible attempts to a reasonable level. At the end of the time limit for each task, the participant was allowed one final round of testing to evaluate their final program's correctness. After each task, we reset the programming environment so that participants could continue with the next task independent of their success on the previous one.

Our primary focus was on Task 1, which we refer to as the CREATE task. This task was designed to test how well participants could apply all the information they gained in their respective training. We asked participants to create a program from scratch that makes the robot pick up an object from the work table and place it into a box. We gave participants 15



**FIGURE 4** Set-up of the experiment: The robot is set up next to the programming environment in front of our participants. In the task shown here, users are instructed to pick up the lego-like object and place it into the white box.

min to solve this task. Since participants had to start from a blank canvas, they were not able to rely on any template or scaffolding besides their training. This allowed us to assume that the quality of training was the determining factor for participants' performance.

After participants solved the first task, we asked them to solve two more tasks which we refer to as the **MODIFY** and **REVERSE** tasks. These tasks allowed us to validate our observations for the **CREATE** task for different task variations. Furthermore, since the tasks took place in sequence, we could observe if differences between participant conditions would diminish over time. For the **MODIFY** task, participants were given an existing, correctly-working pick-and-place program, and asked to change the location where the picked item should be placed. The **REVERSE** task gave participants the same pick-and-place procedure as the **MODIFY** task and asked them to swap the pick and the place location of the task. Both of these tasks were of similar difficulty, but easier than the **CREATE** task. We gave participants 10 min to solve each of these two tasks.

### 4.2.3 | Usability questionnaire

After participants completed all tasks, we asked them to complete a brief survey. This survey contained both a series of demographic questions and a standardized System Usability Scale (SUS) questionnaire<sup>45</sup> The questionnaire asked participants to indicate their agreement with ten statements about the evaluated system on a 5-point Likert scale. The participants' responses allow the computation of an overall SUS score, ranging from 0 to 100 points, that can be compared both between our two participant groups and with the scores of other programming systems.<sup>46</sup> In addition to the standardized questionnaire, we also asked participants to provide written feedback about features of CoBlox that they found particularly easy or difficult to understand, as well as suggestions for improving the training or programming process.

### 4.2.4 | Data analysis

While participants used the CoBlox environment, we collected detailed logs of each user interaction (e.g., adding new blocks, connecting them to existing blocks, defining robot locations). We further collected the final programs created by participants for each task along with video recordings of the robot during the final test run of each program. This way we were able to grade participants' solutions for each task. It also allowed us to analyze which programming steps participants struggled the most with.

We considered both functional correctness and conciseness during grading. We evaluated correctness using a 10-point rubric for each task. Each task was divided into three roughly equally complex sub-tasks (e.g., picking up the item,

transporting it and placing it). We awarded up to 3 points for solving each of these sub-tasks, with partial points given for incomplete or incorrect solutions based on a predefined rubric. We further assigned 1 additional point per task for correctly resetting the robot arm after completing all sub-tasks.

For evaluating conciseness we used a 5-point scale based on the number of used blocks, and penalized programs that used more blocks than necessary. Each additional block compared to the shortest sample solution resulted in a 1-point penalty. We further capped the scores for conciseness relative to the correctness score, so that the scores of short but incorrect solutions would not be inflated.

## 5 | RESULTS

The data we present in this section comes from the 79 participants who completed the full study protocol. Among these participants, we randomly selected 38 (48%) to be trained using the CITs and 41 (52%) to watch a training video. In this section, we compare these two groups with respect to their time spent on training and solving each programming task. We further compare the quality of the programs they wrote as well as their quantitative and qualitative usability feedback. We structure our presentation based on the research questions presented in Section 4.

### 5.1 | RQ1: Time spent on training

All participants that we assigned to the video training condition watched an identical, noninteractive video with a duration of 10 min 51 s. Participants that were trained via CITs were allowed to complete CITs at their own pace, ensuring that they did not have to skip parts of the instructions. All of the participants successfully completed all four of the tutorials, and they took an average of 17 min 1 s (SD 6 m 11 s) to do so.

The majority (68%) of participants completed the CITs within the intended 15-min time frame. However, 12 (32%) participants required more than 15 min to solve all tutorials, and 5 (11%) of them required more than 20 min. We analyzed the 12 participants that took the most time during training more closely. We discovered that almost all of them had substantial technical issues due to a lack of basic technological experience. For instance, in one of these cases, the proctor needed to intervene and demonstrate to the participant how to drag and drop blocks on the used computer.

Overall, the CITs were accessible to end-users, but took longer to complete than watching a training video that covered the same content.

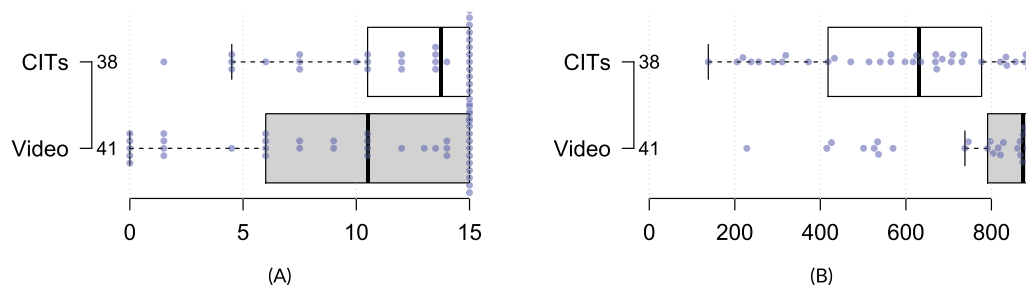
### 5.2 | RQ2: Performance on programming tasks

In the following, we present our quantitative findings on the three programming tasks we asked participants to solve. We start with the results for the CREATE task, which was the first and hardest task, and showed the most substantial difference between the participant groups. Afterwards, we also present the results for the MODIFY and REVERSE tasks, which were similar, but less pronounced.

#### 5.2.1 | The CREATE task

Figure 5 summarizes the quantitative results for the CREATE task. As we described in Section 4, we scored participants' final programs for each task for correctness and conciseness on a scale from 0 to 15 points. The maximum time they for this task was 15 min.

Figure 5A shows that participants trained via the CITs consistently achieved higher scores for the task. The mean score for video subjects is 9.4 (SD 5.7), compared to 12.1 (SD 3.9) for CIT subjects. As Figure 5B shows, participants in the video training condition also consistently needed more time to complete the task. Participants in the video condition spent an average of 13 min 38 s (SD 3 m 24 s) completing their programs. This is substantially longer than the average time of 9 min 59 s (SD 4 m 1 s) that participants trained via the CIT spent on the task.



**FIGURE 5** Box-plots of the quantitative results for the CREATE task. (A) Score (out of 15) for the CREATE task. (B) Time (s) spent on the CREATE task. For all box-plots in this article, center lines show the medians, box limits indicate the 25th and 75th percentiles, whiskers extend to 5th and 95th percentiles, and data points are plotted as open circles.

The time that participants spent on the CREATE task and their resulting score are strongly negatively correlated ( $r = -0.53$ ,  $p < 0.001$ ). The differences between the groups trained via CITs and videos were statistically significant (one-way MANOVA finds  $F(2, 76) = 9.52$ ,  $p < 0.001$ ). We also analyzed times and scores separately and found a statistically significant difference for scores and a significant difference for time (independent samples  $t$ -tests assuming equal variances find:  $t(77) = 4.37$ ,  $p < 0.001$  for time;  $t(77) = -2.44$ ,  $p = 0.017$  for score). The effect size for the times is large ( $d = 0.95$ ) while the effect size for the scores is medium ( $d = 0.54$ ). Notably, only three participants lost points for conciseness, meaning that almost all differences in points are related to correctness.

The results show participants trained via the CITs were substantially faster and more successful at solving the CREATE task than those trained via a conventional training video.

As mentioned in the experimental setup, we limited the time that participants were allowed to spend on each task. This reduced the potential variance between groups by imposing a ceiling on the time-on-task. Thus, in addition to studying the difference between mean times in groups, it is also useful to see how many subjects from each treatment used the maximum time. For the CREATE task, 15 (37%) of the video-trained participants and only 2 (5%) of the tutorial-trained participants required the full 15 min. Furthermore, among those video-trained participants that used all their available time, 11 (73%) scored 0 points. This indicates that these participants were unable to make any significant progress on the task. None of the tutorial-trained participants scored fewer than 2 points on the task.

Our analysis suggests that CITs were effective at helping users who needed support the most.

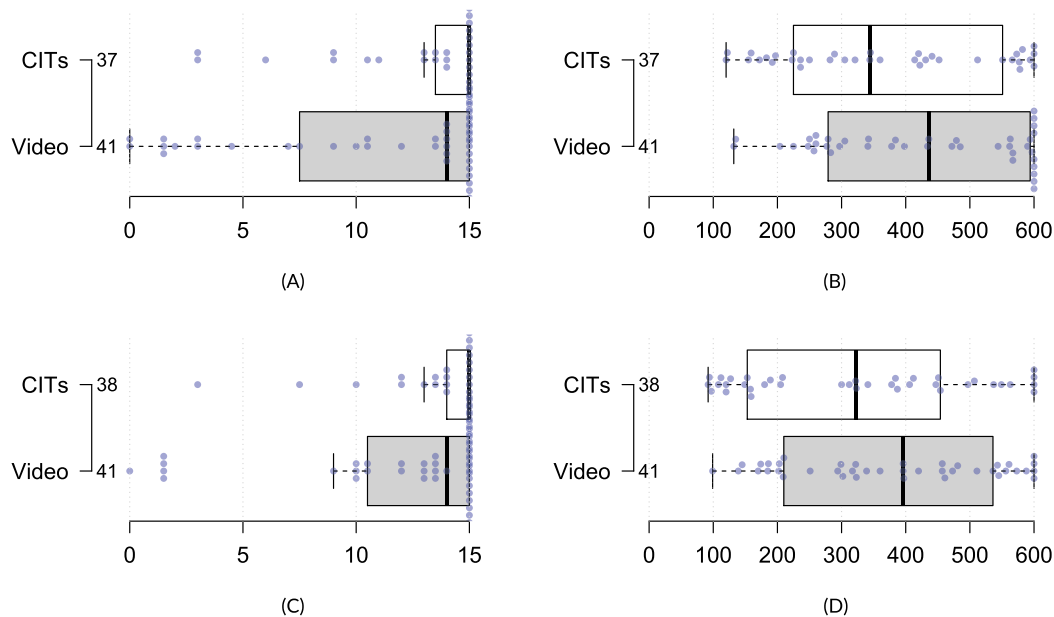
## 5.2.2 | The MODIFY and REVERSE tasks

Figure 6 summarizes the quantitative results for the MODIFY and REVERSE tasks. As for the CREATE task we scored participants' programs on a scale from 0 to 15. The maximum time for these tasks was restricted to 10 min.

For both the MODIFY and the REVERSE task, participants scored higher on average than for the CREATE task. As Figure 6A,C shows, participants trained via CITs scored substantially higher than their video-trained peers on both tasks. For the MODIFY task, CIT-trained participants scored an average of 13.3 points (SD 3.3), compared to their video-trained peers that scored 11.0 points (SD 5.5). The REVERSE task had similar scores, with participants in the CIT condition having an average score of 13.9 points (SD 2.6) compared to 12.1 points (SD 4.9) for the video condition.

As Figure 6B,D shows, participants trained via CITs spent less time to solve both the MODIFY task and the REVERSE task than those trained by video. However, the differences between the two groups were smaller than for the CREATE task, both in absolute numbers and relative to the mean times spent on each task. For the MODIFY task, the mean time was 7 min 20 s (SD 2 m 56 s) for video participants and 6 min 11 s (SD 2 m 48 s) for CIT subjects. For the REVERSE task, the mean time was 6 min 24 s (SD 2 m 50 s) for the video condition and 5 min and 20 s (SD 3 m 1 s) for those trained via CITs.

As for the CREATE task, participant times and scores were negatively correlated for both the MODIFY and the REVERSE task ( $r = -0.56$ ,  $p < 0.001$ ;  $r = -0.38$ ,  $p < 0.001$ ). The difference in times and scores between the participant groups is not statistically significant for either of the two tasks when considering them simultaneously (one-way MANOVA finds



**FIGURE 6** Box-plots of the quantitative results for the MODIFY task and REVERSE task. (A) Score (out of 15) for the MODIFY task. (B) Time (s) spent on the MODIFY task. (C) Score (out of 15) for the REVERSE task. (D) Time (s) spent on the REVERSE task

$F(2, 75) = 2.70$ ,  $p = 0.074$  for the MODIFY task, and  $F(2, 76) = 2.74$ ,  $p = 0.071$  for the REVERSE task). When analyzing times and scores for both tasks separately, we found a statistically significant difference for the scores (independent samples t-tests assuming equal variances find:  $t(76) = -2.29$ ,  $p = 0.025$  for the MODIFY task, and  $t(77) = -2.24$ ,  $p = 0.028$  for the REVERSE task). However, we did not find a significant difference for the times (independent samples t-tests assuming equal variances find:  $t(76) = 1.65$ ,  $p = 0.104$  for the MODIFY task, and  $t(77) = 1.47$ ,  $p = 0.144$  for the REVERSE task). The effect sizes for the scores were large for both tasks ( $d = 0.52$  for the MODIFY task, and  $d = 0.50$  for the REVERSE task) while the effect sizes of the times were medium ( $d = 0.37$  for the MODIFY task, and  $d = 0.34$  for the REVERSE task). Only one participant overall was penalized for a lack of conciseness, by virtually all score differences are based on correctness alone.

The data show that while most participants were able to solve both the MODIFY and the REVERSE task successfully, there was a difference in success between those trained using tutorials and those trained using a conventional training video.

### 5.3 | RQ3: Programming aspects most affected by CIT-based training

To investigate the effects of CITs in more detail and beyond pure performance metrics, we analyzed why the programs that participants wrote achieved sub-optimal scores. For the CREATE task, only 5 (6%) of all programs were penalized for not being concise, while all other penalties were given for correctness issues. For this task, almost all correctness penalties among both groups were given for major errors: 13 (34%) of the CIT-trained and 19 (46%) of the video-trained participants received at least one major penalty for their programs. On closer inspection, most of these major errors were caused by participants struggling to position the robot arm correctly so that it could pick up and carry the item as intended. This type of error, which often led to follow-up errors like dropping the item before it reached its destination, made up 62% of all the major errors in the CIT-trained group and 84% of those in the tutorial-trained group. The programs of 15 (37%) of the video-trained participants missed at least one entire sub-task, compared to 7 (18%) of those trained via CITs.

The data for the CREATE task indicate that CIT-trained participants performed substantially better at positioning the robot arm, which was a programming aspect that was particularly challenging for participants overall.

We also analyzed participants' mistakes for the MODIFY and REVERSE tasks. As for the CREATE task, only a small number of programs (3 for the MODIFY task, 4 for the REVERSE task) were penalized for a lack of conciseness. Since participants performed better on average for these tasks, the number of penalties for missing sub-tasks or major errors was substantially smaller. For the MODIFY task, only 3 (8%) participants in the CIT-trained and 10 (24%) participants in the video-trained group missed an entire sub-task. For the REVERSE task, these numbers were even lower, with only 2 (5%) participants trained via CITs and 5 (12%) trained via video missing an entire sub-task. The percentage of solutions with major errors was also lower for both the MODIFY task (CITs: 11%, video: 24%) and the REVERSE task (CITs: 8%, video: 17%). As for the CREATE task, most major errors were related to issues placing the robot arm correctly to grip the item.

Few participants had issues positioning the physical robot arm for the MODIFY and the REVERSE task, but CIT-trained participants consistently performed better than video-trained ones.

## 5.4 | RQ4: Perceived training quality and usability

We used the System Usability Scale (SUS) questionnaire<sup>45</sup> to measure the perceived usability of CoBlox on a scale from 0 to 100 points, as described in Section 4. The resulting value can be interpreted as a percentile score, with a score of 68 being considered average.<sup>47</sup> We use the adjectives for ranges of scores to ease interpretation, as defined by Bangor et al.<sup>48</sup>

Participants gave the CoBlox programming environment a mean score of 75.6 (SD 11.9). Participants that were trained using the CITs gave CoBlox a mean score of 76.7 (SD 12.2), while participants who were trained using video gave a mean score of 74.5 (SD 11.5). All of these scores fall into the range of "excellent," based on the aforementioned adjective scale.

We conducted a t-test on participants' usability ratings. The difference between these ratings is not statically significant ( $t(77) = 0.82, p = 0.21$ ). We discuss the implications of this finding further in Section 6.3.

The usability of the CoBlox environment was rated highly, regardless of training technique.

In addition to the SUS questionnaire, we also asked participants for what they viewed as the strengths of the environment. 58 participants (73%) responded to this question. Of the responses we received, 36 (20 from the CIT and 16 from the video condition) indicated that the ease of use (or simplicity) of the interface made CoBlox stand out for them. Participants commented on the ease and speed of learning for CoBlox, saying "Even common man can work with this language very quickly" and "Coblox is simple and easy to use. Any layman can use this with a basic guide of 5 min."

Users trained using CITs often commented on their ease-of-use. Four participants that were trained using CITs mentioned that the tutorial system contributed positively to their overall experience. When asked about features that aided them in programming CoBlox, these users clearly identified CITs, saying: "Easy to use blocks, Tutorial" and "User interface and the demo," referring to the tutorial as a demo. In contrast, participants trained using the video commented on known problems with video tutorials,<sup>33</sup> saying "... I was not able to found options as first time I was doing," that we should "have more examples," and that we should have "error messages if user is doing something wrong."

Overall, participants' positive feedback centered around CoBlox ease-of-use features. Some users also specifically mentioned CITs as a helpful feature.

## 6 | DISCUSSION

In this section, we interpret the results of our experimental evaluation and discuss the implications of our work.

### 6.1 | Robot programming: Simulation versus practice

Our study found that for either training method, most adult end-users could quickly learn to use CoBlox and solve simple but realistic robot programming tasks. This finding reinforces the results of a previous, smaller study on CoBlox that used a robot simulator.<sup>12</sup> It also adds to a growing body of evidence that end-users can program professional-grade industrial robots with access to tools that are designed for their specific needs. However, observing the realistic workflow of CoBlox

users also revealed challenges that did not appear in this or other studies that used simulators:<sup>49</sup> participants often had trouble positioning the robot arm precisely enough for their programs to work. Participants also tested their programs much more frequently than in previous studies, since it was harder for them to anticipate the robot's exact behavior. The higher number of test-and-reset cycles also slowed participants down significantly, especially when they were still unfamiliar with the programming environment. These observations highlight the importance of evaluating environments like CoBlox under realistic conditions that provide access both to the programming environment and the physical robot. They also demonstrate the need for additional support when learning end-user programming environments.

Our study demonstrates the advantage of CITs over traditional, video-based training when users have to follow a practical programming workflow. The programming step that users struggled with the most was correctly positioning the robot arm and capturing its coordinates. This observation confirms the assumption that motivated the design of CITs: lead-through programming is difficult to learn for new users and deserves particular attention during training.

While many users struggled with lead-through programming during the tasks they were given, all users successfully completed the physical programming steps that were part of their training via CITs. This indicates that the embedded instructions and immediate feedback that CITs provide were sufficient to support users in performing these steps. Furthermore, while working on the CREATE task, CIT-trained users were significantly more successful applying their learned knowledge in a real task. Not only did they achieve higher scores overall but they also lost fewer points that were specifically related to lead-through programming errors. This indicates that the physical training component of CITs had a positive effect that lasted beyond the completion of the tutorials.

## 6.2 | A place for videos

Our experiment demonstrated that CITs benefit CoBlox users. However, when CITs are used in industrial settings, end-users are only one stakeholder that must be considered: For developers of training systems, CITs add development time and cost compared to recording a training video. This increased investment, if possible at all, must pay off by saving resources or time on training or consistently improving the training outcome. Thus, deciding when to use CITs and when to use less expensive alternatives, such as video training, must not be taken lightly. In our opinion, video tutorials may be best suited for a complementary role to CITs. We see CITs to be particularly helpful in teaching users the basic usage of the programming environment and modes. These training steps are applicable even for other robot models or different usage scenarios.

Once users are familiar with the general programming workflow of CoBlox, training videos could be a cost-effective way to teach a more specialized set of usage scenarios, such as tending specific machines or using model-specific special instructions. Likewise, videos can be personalized in ways that are different than embedded interactive tutorials, such as tutorials that prioritize entertainment (or humor) alongside content or tutorials that situate tasks within a larger conceptual landscape to teach big conceptual topics. It is in the best interest of both users and robot manufacturers to have as many potential resources and avenues of support as possible, be it written manuals, video tutorials, or rich interactive tutorials embedded directly into the system. We therefore see CITs as a relevant addition particularly for the high end robotics market, where investments in training quality can pay off quickly due to productivity improvements and a higher end-user satisfaction.

## 6.3 | Learning versus usability

A surprising result of our study is the lack of correlation between user performance and perceived usability. According to the SUS scores, there was only a small (and not statistically significant) difference between the CIT condition and the video-trained condition, even though there was a significant difference in both time-on-task and correctness of programming solution for the CREATE task. One possible explanation for this could be that we asked users about the perceived usability of the system only at the end of our study protocol. At this point, almost all users were familiar enough with the system to solve the last task in our sequence (REVERSE) successfully. Their answer therefore might have been biased by this recent positive experience.

The primary reason for the higher scores that users achieved in the MODIFY and REVERSE tasks was most likely that they were less challenging. However, the fixed order of the tasks might have been a contributing factor. We suspect that particularly for those users trained by video, the first task they worked on acted as a self-administered tutorial. This would

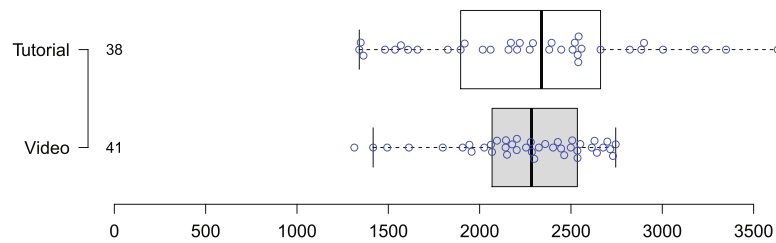


FIGURE 7 Box-plot of the total time (s) spent by participants on both training and tasks

explain why video-trained users performed worse and took significantly longer for this task, but the difference faded as the study progressed. It would also demonstrate how a beginner-friendly design can help overcome differences in training quality, as long as users are resilient enough to teach themselves how to use the system.

## 6.4 | Training and completion times

Our results show that while participants trained using CITs solved tasks significantly faster, they also spent significantly more time on training. This suggests that the benefits of tutorials decrease when training time is taken to consideration. As Figure 7 shows, the average sums of training and completion times are similar for each treatment. Nonetheless, these data deserve a closer look.

As Figure 7 shows, the overall completion times of participants trained by tutorials show a wider spread than those of the video-trained participants. We attribute the wider range of overall completion times for tutorial-trained participants to two factors: First, due to business constraints, we had to impose a strict limit on the time spent on each task. This artificially limited the overall time for participants that used the maximum time allocated for each task. The vast majority (87%) of these participants were in the video-trained group. Had these participants been given unlimited time to work on the task, the average time for video-trained users to complete tasks would have likely been higher, and in some cases much higher; nine video-trained participants scored less than five out of fifteen on the first task, indicating they had made little progress prior to the time limit.

Second, the wider spread might also indicate that CITs gave participants more freedom to take the time that they needed for their training instead of having to keep up with the fixed pace of the training video. We allowed participants to rewind the video and also to freely try out the CoBlox environment before they started working on the given tasks, but almost none of them made use of this opportunity. We interpret this as evidence that the additional guidance that the tutorials provided has helped participants to self-pace and not overestimate their understanding of the system. However, an explicit prompt for participants to familiarize themselves with CoBlox, for example by reproducing what they had seen in the video, might have led to a more similar distribution of training times between the groups and also removed this as a source of bias in our study results.

## 6.5 | Threats to validity

*External validity.* With respect to generalizability, our evaluation of CITs took place at a single industrial engineering firm, and all participants were employees at that firm. Additionally, the CIT system was tailored for the CoBlox programming language and one specific model of industrial robot. The tasks we have chosen further only cover the creation and modification of code, while real tasks that end-users face in practice may be more diverse and cover other software engineering aspects. For example, we have not investigated whether end-users would be capable of ensuring that code is robust or maintainable, and it is not clear whether CITs could provide substantial benefits when considering this type of task.

*Internal validity.* In terms of study design, one major limitation was the artificial time constraints for the programming tasks. While this was absolutely necessary because the study was conducted as part of a normal industrial workday, it truncated the programming task time for some participants, leading to artificially low task completion times. Had participants been able to work on tasks as long as needed for completion, we believe the tasks times for video-trained participants would have been much higher, as many video-trained participants were far from finishing at the end of the allotted time.



*Construct validity.* Another limitation is the training method we have chosen to compare CITs against in our study. We have selected video-based training since this type of training is popular in practice<sup>44</sup> and the necessary materials were easy to create in the context of our study. However, videos alone are not typically considered to be a form of intelligent tutoring since they are not interactive. Participants had a chance to explore the CoBlox environment by themselves, but were not explicitly told to do so before starting to work on our experimental tasks. It is therefore unclear how much influence their familiarity with CoBlox had on our study results. Our study can also not answer whether other types of tutoring systems, such as a disconnected or coexisting tutorials, would have led to comparable improvements in the participants' performance.

Due to the fixed task order, participants' performance on later tasks might have also been affected by them getting more familiar with the system as they worked on tasks. Video-trained participants in particular might have used more of their time on the first task to explore the system. This might have lowered their scores for that task, but improved their scores for later tasks as they gained familiarity with the system.

## 7 | CONCLUSION

Robotics systems will continue to increase in numbers as their costs decrease. Concurrently, the fraction of the costs associated with programming these systems will continue to grow. To address this, end-user industrial engineers, who are not trained software developers, need to learn to effectively program these systems. In this article, we demonstrated that CITs have the ability to scaffold industrial end-users as they learn to create robotics programs. Through an experiment with 79 industrial end-users at a single industrial corporation, we compared the effectiveness of CITs to video-based training which has been historically used for learning these systems. This evaluation showed that industrial end-users trained with CITs were both quicker and more effective than those trained with video-based interventions. While interactive tutorials may be more expensive than video training, this result demonstrates that these costs could be worth bearing in cost-sensitive domains.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thorough and insightful feedback. This work has been funded by the National Science Foundation through grant 2024561, by NSERC through a Collaborative Research and Development Grant, and via an ABB Research grant.

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

## AUTHOR CONTRIBUTIONS

Nico Ritschel performed the majority of the analysis and interpretation of the experimental results, created the majority of figures found in the article, and lead the initial writing and later revisions of the article. Anand Ashok Sawant designed and implemented the CIT system, and contributed to the qualitative evaluation and the experimental design, as well as the initial writing of the article. David Weintrop supervised and contributed to the design of CITs and contributed to the discussion of related work. Reid Holmes is a co-supervisor of Nico Ritschel and has provided feedback on the experimental evaluation, its presentation and has contributed to the writing of the article. Alberto Bacchelli is the supervisor of Anand Ashok Sawant and has provided feedback on the experimental design and the presentation of the qualitative results. Ronald Garcia is a co-supervisor of Nico Ritschel and has provided feedback on the experimental evaluation, the statistical analysis and the writing and structure of the article. Chandrika K R and Avijit Mandal have supervised and proctored the experiments and collected the raw data for the evaluation. Patrick Francis has supported the development and implementation of CITs and provided feedback and details for the description of their features. David C. Shepherd has supervised and coordinated the research project, provided feedback on all aspects of the design of CITs and their evaluation, and has contributed to the writing and revision of the article.

## ORCID

Nico Ritschel  <https://orcid.org/0000-0001-5600-2978>

Anand Ashok Sawant  <https://orcid.org/0000-0002-5816-8020>

## REFERENCES

1. Kock S, Vittor T, Matthias B, et al. Robot concept for scalable, flexible assembly automation: a technology study on a harmless dual-armed robot. *Proceedings of the International Symposium on Assembly and Manufacturing (ISAM)*; 2011:1-5.
2. Hagele M. Robots conquer the world [turning point]. *IEEE Robot Autom*. 2016;23(1):120-118.
3. Pan Z, Polden J, Larkin N, Van Duin S, Norrish J. Recent progress on programming methods for industrial robots. *Proceedings of ISR and ROBOTIK*. Springer; 2010:1-8.
4. Biggs G, MacDonald B. A survey of robot programming systems. *Proceedings of ARAA*. Springer; 2003:1-10.
5. Ajaykumar G, Steele M, Huang CM. A survey on end-user robot programming. arXiv preprint arXiv:210501757; 2021.
6. Bischoff R, Kazi A, Seyfarth M. The MORPHA style guide for icon-based programming. *Proceedings of RO-MAN*. Springer; 2002:482-487.
7. Lego Systems Inc. Lego mindstorms NXT-G invention system; 2008.
8. Ritschel N, Kovalenko V, Holmes R, Garcia R, Shepherd DC. Comparing block-based programming models for two-armed robots. *IEEE Trans Softw Eng*. 2022;48(5):1630-1643.
9. Kushida D, Nakamura M, Goto S, Kyura N. Human direct teaching of industrial articulated robot arms based on force-free control. *Artif Life Robot*. 2001;5(1):26-32.
10. Sefidgar YS, Weng T, Harvey H, Elliott S, Cakmak M. RobotIST: interactive situated tangible robot programming. *Proceedings of the Symposium on Spatial User Interaction*; 2018:141-149.
11. Cao Y, Wang T, Qian X, et al. GhostAR: a time-space editor for embodied authoring of human-robot collaborative task with augmented reality. *Proceedings of UIST*. Springer; 2019:521-534.
12. Weintrop D, Afzal A, Salac J, et al. Evaluating CoBloX: a comparative study of robotics programming environments for adult novices. *Proceedings of SIGCHI*. Springer; 2018:366:1-366:12.
13. Dorn B, Guzdial M. Learning on the job: characterizing the programming knowledge and learning strategies of web designers. *Proceedings of SIGCHI*. Springer; 2010:703-712.
14. Guo PJ. Older adults learning computer programming: motivations, frustrations, and design opportunities. *Proceedings of SIGCHI*. Springer; 2017:7070-7083.
15. Soloway E, Guzdial M, Hay KE. Learner-centered design: the challenge for HCI in the 21st century. *Interactions*. 1994;1(2):36-48.
16. Cooper L, Orrell J, Bowden M. *Work Integrated Learning: A Guide to Effective Practice*. Routledge; 2010.
17. Bottani E, Vignali G. Augmented reality technology in the manufacturing industry: a review of the last decade. *IISE Trans*. 2019;51(3):284-310.
18. Grossman T, Fitzmaurice G. ToolClips: an investigation of contextual video assistance for functionality understanding. *Proceedings of SIGCHI*. Springer; 2010:1515-1524.
19. Huang J, Twidale MB. Graphstract: minimal graphical help for computers. *Proceedings of UIST*. Springer; 2007:203-212.
20. Kelleher C, Pausch R. Stencils-based tutorials: design and evaluation. *Proceedings of SIGCHI ACM*. Springer; 2005:541-550.
21. Wang CY, Chu WC, Chen HR, Hsu CY, Chen MY. EverTutor: automatically creating interactive guided tutorials on smartphones by user demonstration. *Proceedings of SIGCHI*. Springer; 2014:4027-4036.
22. Kim Y, Choi Y, Kang D, Lee M, Nam TJ, Bianchi A. Heyteddy: conversational test-driven development for physical computing. *Proc UBICOMP*. 2019;3(4):1-21.
23. Warner J, Lafreniere B, Fitzmaurice G, Grossman T. ElectroTutor: test-driven physical computing tutorials. *Proceedings of UIST*. Springer; 2018:435-446.
24. Carroll JM, Rosson MB. *Paradox of the Active User*. MIT Press; 1987.
25. Grossman T, Fitzmaurice G, Attar R. A survey of software learnability: metrics, methodologies and guidelines. *Proceedings of SIGCHI*. Springer; 2009:649-658.
26. Bloom BS. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educ Res*. 1984;13(6):4-16.
27. Corbett A. Cognitive computer tutors: solving the two-sigma problem. In: Bauer M, Gmytrasiewicz PJ, Vassileva J, eds. *User Modeling*. Springer; 2001.
28. VanLehn K. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educ Psychol*. 2011;46(4):197-221.
29. Anderson JR, Boyle CF, Reiser BJ. Intelligent tutoring systems. *Science(Washington)*. 1985;228(4698):456-462.
30. Kim AS, Ko A. A pedagogical analysis of online coding tutorials. *Proceedings of SIGCSE*. Springer; 2017:321-326.
31. Ashenfelter E. From the classroom: a guide to teaching coding using Google's CS first. *Gift Child Today*. 2017;40(4):220-225.
32. NI Learning Center. *LabVIEW Tutorial: Getting Started with LabVIEW Programming Basics*. NI Learning Center; 2022 <https://learn.ni.com/learn/article/labview-tutorial>
33. Knabe K. Apple guide: a case study in user-aided design of online help. *Proceedings of the Conference Companion on Human Factors in Computing Systems*; 1995:286-287.
34. Code Inc. Hour of code. CoDesign; 2020. <http://code.org/learn>.
35. Resnick M, Silverman B, Kafai Y, et al. Scratch: programming for all. *Commun ACM*. 2009;52(11):60.
36. Cooper S, Dann W, Pausch R. Alice: a 3-D tool for introductory programming concepts. *J Comput Sci Coll*. 2000;15(5):107-116.
37. Bergman L, Castelli V, Lau T, Oblinger D. DocWizards: a system for authoring follow-me documentation wizards. *Proceedings of UIST*. Springer; 2005:191-200.
38. Fernquist J, Grossman T, Fitzmaurice G. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. *Proceedings of UIST*. Springer; 2011:373-382.

39. Metcalfe J. Learning from errors. *Annu Rev Psychol.* 2017;68:465-489.
40. Barobo Inc. *RoboBlockly*. Barobo Inc.; 2020 <https://roboblockly.com>
41. Li W, Grossman T, Fitzmaurice G. GamiCAD: a gamified tutorial system for first time autocad users. Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology; 2012:103-112.
42. Davis JU, Gong J, Sun Y, Chilana P, Yang XD. CircuitStyle: a system for peripherally reinforcing best practices in hardware computing. *Proceedings of UIST*. Springer; 2019:109-120.
43. Metcalfe J. Learning from errors grantee submission. *Psychology.* 2017;68:465-489.
44. van der Meij H, Van Der Meij J. A comparison of paper-based and video tutorials for software learning. *Comput Educ.* 2014;78:150-159.
45. Brooke J. SUS-A quick and dirty usability scale. *Usab Eval Ind.* 1996;189(194):4-7.
46. Bangor A, Kortum PT, Miller JT. An empirical evaluation of the system usability scale. *J HCI.* 2008;24(6):574-594.
47. Lewis JR, Sauro J. The factor structure of the system usability scale. Proceedings of the International Conference on Human Centered Design; 2009:94-103.
48. Bangor A, Kortum P, Miller J. Determining what individual SUS scores mean: adding an adjective rating scale. *J Usab Stud.* 2009;4(3):114-123.
49. Weintrop D, Shepherd DC, Francis P, Franklin D. Blockly goes to work: block-based programming for industrial robots. Proceedings of Blocks and Beyond Workshop (B&B); 2017:29-36.

**How to cite this article:** Ritschel N, Sawant AA, Weintrop D, et al. Training industrial end-user programmers with interactive tutorials. *Softw Pract Exper.* 2022;1-19. doi: 10.1002/spe.3167