



# Mapping aids using output-directed programming increase novices' performance in programming mobile robotic systems

Thomas Witte<sup>1</sup> · Andrea Vogt<sup>2</sup> · Tina Seufert<sup>3</sup> · Matthias Tichy<sup>1</sup> 

Received: 26 November 2024 / Accepted: 24 August 2025  
© The Author(s) 2025

## Abstract

**Context:** Novices programming robotic systems' behavior, like quadcopter missions, face several challenges and require adequate support to overcome initial barriers. One approach to support novices is to display multiple representations such as graphical previews along with the code editor. Such supportive representations, however, also pose challenges for novices: finding corresponding information in the code and in the preview. To facilitate this, mapping aids can be implemented to clarify the connections between code and preview and foster a deeper understanding. Using output-directed programming, that is, adding the ability to reverse expression evaluation in the domain-specific language, is a promising basis for easily creating and implementing mapping aids.

**Objective:** We investigated, whether mapping aids based on output-directed programming can improve learning language semantics and overall program correctness and how these mapping aids support novices while implementing quadcopter missions.

**Method:** In our study, we tested  $N = 82$  participants while interacting and learning in an online programming environment. Using our 2x2 between-subject design study, we investigated the effects of two mapping aids: highlighting (supports to find element-based connections in the environment) and dynamic linking (supports finding similarities on the semantic level of the content) on program correctness including a typical error, learning outcomes as well as traces of learning strategies.

**Results:** While highlights were more helpful for implementing the quadcopter missions (mission 1:  $p = .008^{**}$ ,  $\eta^2_{\text{partial}} = .091$ ), dynamic linking improved learning outcomes on the comprehension ( $F(1, 75) = 5.61$ ,  $p = .020^*$ ,  $\eta^2_{\text{partial}} = .070$ ) and application level ( $F(1, 75) = 4.08$ ,  $p = .047^*$ ,  $\eta^2_{\text{partial}} = .052$ ). Traces of learning strategies were related to higher program correctness (organizing (changes in the preview)):  $r = .553$ ,  $p < .001^{***}$ ; elaborating (time engaging in the task)):  $r = .639$   $p < .001^{***}$ ) and higher learning outcomes (organizing:  $r = .400$ ,  $p < .001^{***}$ ; elaborating :  $r = .404$ ,  $p < .001^{***}$ ).

---

Communicated by: Scott Fleming.

Thomas Witte and Andrea Vogt contributed equally to this work

---

Extended author information available on the last page of the article

**Conclusions:** Implementing mapping aids through output-directed programming supports novices in developing a better semantic understanding of the domain specific language. Depending on the program tasks, different mapping aids might be effective. Based on traces of learning strategies while programming, adaptive interactive programming environments might support users individually.

**Keywords** Domain specific languages · Robotic systems · Novices · Output-directed programming · Provenance tracking · Bidirectional linking · Multiple representations · Mapping aids · Traces of learning strategies

## 1 Introduction

The programming and integration of stationary or mobile robots is often performed by domain experts rather than software engineers (Rossano et al. 2013). For example, drones can assist in rescue operations by improving accident assessment, simplifying coordination, or contributing to the rescue of survivors (Roldán-Gómez et al. 2021). The low-level programming of such a system, i.e., basic control and communication capabilities, is provided by the manufacturer. However, high-level programming of the mission objectives must be done on-site by the drone operator – often a programming novice instead of a programming expert.

Novice programmers, i.e., users without or with very little prior programming knowledge or experience in algorithmic thinking (cf. Section 2.1), often face multiple challenges simultaneously. For example, they are concurrently challenged by the syntactic and semantic understanding of the language, as well as by algorithmic problem solving (Qian and Lehman 2017). Hence, programming novices must be supported in gaining a basic understanding of the language while learning to program the robotic system. For this purpose, simplified domain-specific languages (DSLs) and targeted programming and learning environments (PLEs) exist for these systems, such as the *iRobot root coding app*<sup>1</sup>, or *CoBlox* (Weintrop et al. 2018).

DSLs can be tailored to the domain and provide matching abstractions to reduce program complexity. PLEs therefore often use DSLs and provide additional visualizations to further reduce complexity for novice programmers. To link or map these visualizations to matching pieces of code, the language (tooling) must provide additional introspection capabilities, such as exposing code locations or dependency information. Hence, the more information is provided by default in the DSL (e.g. about the dataflow), the easier it is to create mapping aids in the PLE. These mapping aids (e.g. Gentner 1983; Brünken et al. 2005; Fries et al. 2021) can then help novice users relate multiple representations, such as a coding editor with a code preview (Fries et al. 2021; Seufert 2019; van der Meij and de Jong 2006) and facilitate the understanding of the corresponding processes.

Output-directed programming (Hempel et al. 2019; Witte and Tichy 2019) links the result of a computation to its inputs, making algorithmic black boxes more transparent and explainable to users. It can be implemented by tracking the data-flow at run-time and adding provenance metadata to values. This metadata can then be queried for input values that produce a desired output value, i.e., to answer the question “How do I need to change the input

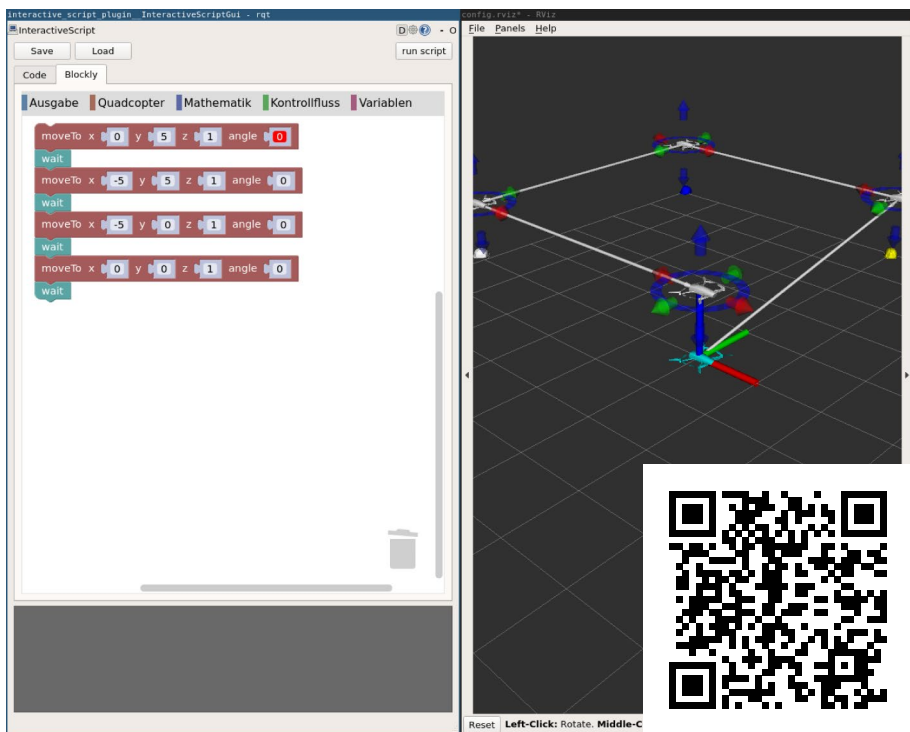
<sup>1</sup> <https://edu.irobot.com/what-we-offer/irobot-coding>

to produce the result  $X$  instead?". Output-directed programming thus provides the necessary information to implement mapping aids for novices.

To address the aforementioned problems programming novices face, we developed a PLE for mobile robot missions with a live preview of the planned flight path and output-directed programming capabilities (Witte and Tichy 2019), shown in Fig. 1. The code editor uses Blockly on top of a minimal DSL to minimize syntactic complexity (Pasternak et al. 2017; Price and Barnes 2015). Two mapping aids use the output-directed programming capabilities of the DSL's runtime to help relate code blocks with their corresponding preview: 1) a *highlighting* feature allows clicking waypoints in the preview and highlights all value/number blocks that influence the position of the waypoint, and 2) a *dynamic linking* feature to directly move waypoints in the preview and automatically change the code accordingly.

A beneficial effect for novice programmers was expected as the implemented mapping aids support mapping the multiple representations and make dependencies and interrelations more obvious. The implemented mapping aids were selected as examples of *element-to-element* and *relation-to-relation* mapping (Gentner 1983) as explained in Section 2.2. The semantic knowledge to power these mapping aids is collected at run-time by tracking data provenance.

In particular, we investigated whether these mapping aids support novice programmers in learning the semantics of a domain-specific language for a mobile robot. We focus not



**Fig. 1** PLE with output-directed programming: moving waypoints in the preview reverses the evaluation and changes the program accordingly. Video demo: [https://github.com/sp-uulm/interactive\\_script/wiki/Interactive-Script-Demo](https://github.com/sp-uulm/interactive_script/wiki/Interactive-Script-Demo)

only on the semantic correctness of the resulting program, but also on the deeper learning processes while using our PLE.

During PLE interaction, learners use common strategies, such as organizing and elaborating, as part of their learning that hint at specific cognitive and metacognitive processes. Process data, i.e., screenshots, save states, can be collected and analyzed for traces of such learning strategies related to programming (Hadwin 2021; Jeske et al. 2014). We wanted to further investigate how these learning strategy traces relate to the program correctness and the learning outcomes.

In detail, our study aimed to answer the following research questions (RQs) in the context of our PLE for novice programmers:

*RQ1* How effective are provenance-based mapping aids in increasing (semantic) program correctness?

*RQ2* How effective are provenance-based mapping aids in improving learning outcomes?

*RQ3* How are the learning strategy traces and the program correctness related?

*RQ4* How are the learning strategy traces and the learning outcome related?

*RQ5* What is a typical error and how do these mapping aids help novices reduce these errors?

We conducted a 2x2 between-subject design study with 82 participants to gain insights into these different mapping aids (highlighting and dynamic linking) and their impact on performance (program correctness and learning outcome) under consideration of participants' prior knowledge, figural intelligence, and need for cognition. We selectively recruited participants, who match our notion of programming novices and referred to these participants as novices throughout the paper.

Each participant first completed a questionnaire to measure likely influencing factors such as prior knowledge, figural intelligence, and need for cognition. They then remotely connected to the PLE to complete three novice-oriented programming tasks – creating flight missions for a simulated quadcopter in 3D space. Finally, a second questionnaire was used to assess learning outcome and cognitive load while using the PLE.

By implementing this multi-method approach, we investigated the effects of mapping aids on the program correctness of three different programming tasks (quadcopter missions; RQ1) as well as on learning outcomes (knowledge, comprehension, and application level; RQ2). Based on screenshots and saved data, we examined the effects of mapping aids on the traces of learning strategies (RQ3 & RQ4) and a typical error during programming (RQ5).

Based on our findings for RQ1 and RQ2, the highlighting mapping aid enabled novices to achieve better program correctness (mission 1:  $p = .008^*$ ,  $\eta^2_{\text{partial}} = .091$ ) while the dynamic linking mapping aid increased the learning outcome (comprehension:  $F(1, 75) = 5.61$ ,  $p = .020^*$ ,  $\eta^2_{\text{partial}} = .070$  & application:  $F(1, 75) = 4.08$ ,  $p = .047^*$ ,  $\eta^2_{\text{partial}} = .052$ ). Overall, the best performance was found when both mapping aids were available. Having both mapping aids leads to a higher learning outcome than just the highlighting aid ( $p_{\text{application}} = .013^*$ ). Program correctness with both mapping aids is higher than with just dynamic linking for the first and the second mission ( $p_{\text{mission1}} = .002^*$ ,  $p_{\text{mission2}} = .015^*$ ); and higher than with just highlighting in the second mission ( $p_{\text{mission2}} = .042^*$ ). For RQ3 and RQ4, traces of learning strategies (organizing, elaborating) were positively related to both the program correctness and the learning outcome. For organizing, correlations were found between perspective changes in the preview and pro-



gram correctness ( $r = .553, p < .001^{***}$ ), as well as learning outcome ( $r = .400, p < .001^{***}$ ). For elaborating, correlations were found between task time and program correctness ( $r = .639, p < .001^{***}$ ) as well as learning outcome ( $r = .404, p < .001^{***}$ ). Participants in the group with both aids had a 5.18-times higher chance ( $p = .020^*$ ) of avoiding a certain typical implementation mistake (RQ5).

These results indicate that a combination of multiple mapping aids based on output-directed programming can improve both program correctness and learning outcomes on different levels (knowledge, comprehension or application). Based on our findings, the quality of the provenance traces generated by our PLE is sufficient to increase the semantic understanding of the DSL and the environment. Furthermore, our findings imply that the presentation of additional information in the PLE, such as provenance data in the form of mapping aids, could have a significant influence on the underlying cognitive or psychological processes and the learning strategies used.

The remainder of this paper is structured as follows: Section 2 summarizes the technical and psychological background and presents the PLE used in the study. The hypotheses derived from the related literature and research questions are given in Section 3. In Section 4, we describe the study design, methods, and procedures in more detail. The results of our study are then presented in Section 5 before discussing the implications of these results, as well as threats to validity and related work in Sections 6 and 7. In Section 8, we conclude and summarize our results and give a more detailed outlook on possible future work.

## 2 Background

In the following, we provide some in-depth insight into the background of our work to make it accessible to readers from different communities. We describe the characteristics of *novices* in Section 2.1, give an overview of their cognitive processes while learning programming in Section 2.2, and how the learning outcome can be assessed (Section 2.3). Furthermore, the PLE, as the object of our study and its help features that support novice programmers, is introduced. We describe the technical foundation of our PLE in Section 2.4, the PLE itself, as it is used in our study, and the technical details of how the two mapping aids, the independent variables of our study, are implemented in Section 2.5.

### 2.1 Novice Programmers

There is no single definition of *novice programmers* in the literature: In most cases (e.g., Lister et al. 2004; Vainio and Sajaniemi 2007; Lahtinen 2007) programmers are considered novices if they attended up to one semester of programming courses or only have a basic understanding of programming. Fisher (1991) recommends clearly stating the characteristics of novices to better compare studies as terms such as *novice* or *naïve* are not sufficient to describe the multifaceted skills and knowledge of these users. Mayer (1981) defines novices as “*users who have had little or no previous experience with computers, ..., and who thus lack specific knowledge of computer programming.*” (Mayer 1981, p.123).

In line with this definition, our study targets novice programmers in their effort to gain a basic understanding of the semantics of our programming environment and domain-specific language but not become professional programmers. Therefore, we consider a person a

programming novice if they 1) did not attend programming courses or learned to program for an extended period of time, 2) did not use similar programming environments before, and 3) used our environment only for the duration of the study. Furthermore, we measure and correct for differences in prior knowledge in participants in our study as detailed in Section 4.5.

Qian and Lehman (2017) conducted a literature review on the challenges novice programmers face: these include syntactic problems, insufficient mental models of language semantics, as well as problems in problem solving or applying general programming and debugging strategies. Among these problems in semantic understanding, tracing, i.e., simulating program execution and results in one's head, is commonly examined and identified as an ability that separates expert from novice users (e.g., Lister et al. 2004; Vainio and Sajaniemi 2007; Venables et al. 2009).

Block-based code representations, such as Blockly (Pasternak et al. 2017), are commonly used to help novice programmers achieve syntactically correct programs. Compared to textual interfaces, block-based robot programming interfaces improve *learnability* of the environment and reduce *time on task* with similar correctness of the result in a study with 67 (adult) novices by Weintrop et al. (2018). However, the authors of the study note that the participants still had problems correctly positioning and manipulating the robot arm, advocating for a better integration of the 3D preview with the program editor.

## 2.2 Cognitive Processes in Working with Multiple Representations

Multiple representations of code, e.g., as block-based code elements in the editor and as 3D-previews of a flight trajectory, are commonly used to alleviate one or more of these common problems novices face. When multiple representations are included in a PLE, they can have different roles (Ainsworth 2014): 1) complement each other, 2) limit each other's interpretation, or 3) promote deeper understanding.

In the case of interactive PLEs, i.e., PLEs that the learner can interact with or manipulate directly, multiple representations often take the third role, with the goal of fostering a deeper understanding (Rey 2011). To facilitate this, additional *mapping aids* are provided in such PLEs to help locate the corresponding information and thus simplify the mapping (e.g. Fries et al. 2021; van der Meij and de Jong 2006). Mapping processes can be distinguished into element-to-element and relation-to-relation mapping processes (Gentner 1983; Seufert 2019). The term element-to-element mapping refers to element-based connection, reflecting that novices are able to connect different components on a rather superficial level (Fries et al. 2021; Gentner 1983). In contrast, the term relation-to-relation mapping refers to the finding of similarities on the semantic level and thus requires novices to have a deeper understanding of the elements and processes that need to be mapped and integrated (Patwardhan and Murthy 2017; Gentner 1983; Seufert 2019).

We evaluated two forms of mapping aids that help novices in our PLE: 1) *highlighting* values that affect the selected output value as a form of element-to-element mapping enabled our participants to click on a waypoint to highlight all literal values that may influence its position, and 2) *dynamic linking* as a form of relation-to-relation mapping, which allowed participants to directly move waypoints through the program preview.

If enough cognitive resources are available after relating these different representations, processing and learning is successful and new information is integrated and linked in a

complex, analog *mental model* (Schnotz and Bannert 2003). However, working memory might be (over-)loaded by the presentation of many different pieces of information at the same time, where the connections between these pieces remain unclear (van Someren et al. 1998). In particular, learners with little prior knowledge cannot manage working memory capacity by chunking and are particularly affected by this problem (Kozma and Russell 1997). Furthermore, these challenges could cause novices to use mapping aids in a disorganized and trial-and-error manner rather than a methodical manner (Patwardhan and Murthy (2017); de Jong and van Joolingen (1998)).

## 2.3 Performance Assessment and Learning Strategy Traces

Different cognitive sub-steps while interacting and learning in a PLE can be distinguished and might only become apparent when using a differentiated measurement of not only task performance but also learning outcome.

The performance of novices handling programming tasks (program correctness; see Appendix C) can be used as a criterion to evaluate how novices can be successfully supported by mapping aids (RQ1). In order to solve the tasks successfully and achieve good performance in program correctness, novices must develop a basic understanding of the language and interactions with the environment. Based on Bloom et al. (1956) the three levels of learning outcome, knowledge, comprehension, and application, can be distinguished (RQ2). The *knowledge* level can be achieved simply by storing a superficial representation of the learning content. In contrast, users need to semantically process the information to succeed at the *comprehension* and the *application* level (Sobral 2021). More specifically, the ability to contrast different concepts or code fragments and to understand the relationship between two code elements represents the learning outcome at the *comprehension* level. When users have integrated the new information into the more global and analog mental model, they can deduce more complex consequences and interactions between the different components, as well as *apply* the concepts to different problems (Mayer et al. 2002).

Users employ learning strategies, e.g., *organizing or elaboration*, that influence how they process information (Weinstein and Underwood 1985; Mayer 1988). These strategies can have many forms, such as note-taking, planning, or reviewing information. However, self-assessment of these learning strategies is difficult and often unreliable (Weinstein and Underwood 1985; Winne and Jamieson-Noel 2002). Cognitive learning strategies such as organizing and elaborating can be derived based on learning behavior from process data (Hadwin 2021; Jeske et al. 2014). Hadwin et al. (2007) uses clustering and frequency analysis to automatically scan the recorded process data, such as log files, for specific signs or traces of these learning strategies. In Fincham et al. (2018); Matcha et al. (2019), Hidden Markov Models (HMM) are used to identify different learning strategies in multiple streams of process data.

In addition, learning behavior can contain traces of metacognitive strategies, which can map, for example, planning or monitoring during the programming process (Hadwin 2021; Jeske et al. 2014; Wild and Schiefele 1994). Hence, traces of certain strategies while working on programming tasks could already provide information about whether users use adequate strategies and whether they achieve a higher level of performance with these strategies (RQ3 & RQ4). Furthermore, an example of a typical error can be identified that provides additional insight on the effect of the mapping aids (RQ5).

## 2.4 Provenance Tracking and Output-Directed Programming

Programming environments for novices often provide a preview of the code by repeatedly executing or simulating the code as it is edited. Using a visualization in the form of a preview might foster program understanding (Tanimoto 2013). This graphical output or output that has a spatial or physical dimension, e.g., visualizations of the environment, is well suited for the aim of *liveness*, i.e., allowing direct user interaction with the running program either by an automatic feedback loop or by manually starting a simulation.

Robot programming can profit from live programming, as the feedback loops when testing new code are especially long: transferring and running programs on the robot and observing the results is often time-consuming and might even pose safety risks to the operator (Campusano and Fabry 2017). In contrast, a live preview provides immediate feedback and might even incorporate data from the physical environment or a simulation to help identify potential errors or safety hazards. In addition, live programming features help to form mental models of the program as well as language semantics and can lead to more experimentation (Kang and Guo 2017).

Provenance traces can be used to create bidirectional linked representations, i.e., changes to the preview that are mapped to changes to the code that produce the desired result. The evaluation of the program can be reconstructed and reversed based on the recording of the data provenance of values (Acar et al. 2013), i.e., the origin and history of values in a program, while rendering the preview. This is called *Output-directed programming* (Hempel et al. 2019; Witte and Tichy 2019). This process is mostly application- and domain-independent. It was previously applied, for example, in drawing programs (Hempel et al. 2019), live evaluation in text editors (Breckel and Tichy 2016; Hempel and Chugh 2020), HTML and Markdown editors (Mayer et al. 2018). Due to tracking data flow during execution and zero domain knowledge, output-directed programming can be easily integrated into existing live programming environments but introduces some runtime overhead. Compared to static analysis approaches, provenance tracking can produce unexpected results in the presence of aliasing or changes in the dataflow.

Specifically, provenance tracking and output-directed programming can be used to implement multiple linked representations of code in a straightforward way (*dynamic linking* in our PLE) or identify input values that influence the result (*highlighting*). Together, the linked graphical preview and interaction features based on output-directed programming form the basis of a live programming environment that helps novices by explaining relations or helping them interpret results.

## 2.5 Study Programming and Learning Environment

For our study, we adapt the programming environment by Witte and Tichy (2019) that implements two mapping aids based on output-directed programming: a block-based code editor is displayed alongside a graphical preview of the program (Fig. 1). The code editor uses a Lua-based DSL that is designed around a strictly imperative high-level programming paradigm to program quadcopter missions using only a few basic functions. To further help novice programmers, a *Blockly*-based visual program editor reduces the syntactic complexity of the language and encourages experimentation (Kölling et al. 2017). Each basic function

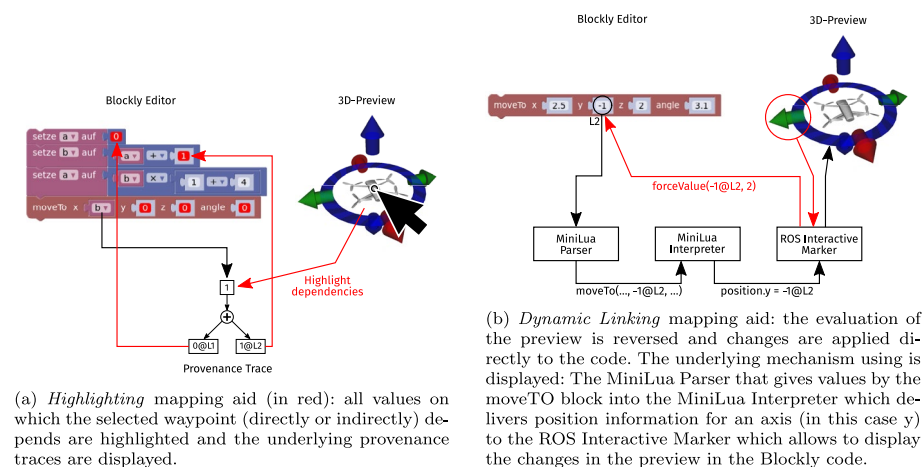
is represented by a puzzle block; connecting it to other pieces creates a syntactically correct program. A live visualization of the currently edited program is shown in a 3D preview in real-time, displaying waypoints and the flight path of the quadcopter. The interactive preview shows all waypoints and flight paths ignoring the time dimension. However, users can start a simulation of the quadcopter that shows an animated 3D model of the quadcopter executing the mission in real-time.

The language interpreter implicitly tracks the provenance of each value and provides the data for two mapping aids based on output-directed programming. If a quadcopter is instructed to fly to a certain waypoint using a *moveTo* block, as shown in Fig. 2a, the code locations/blocks and operations defining the quadcopter's position along the *x*-Axis are known throughout the complete execution and rendering process of the preview.

Based on provenance data, two mapping aids that help novices relate both representations were implemented:

**Highlighting** First, our PLE integrates a *Brushing* (Becker and Cleveland 1987) style mapping aid: selecting a waypoint in the preview highlights the corresponding values in the code. Although there is a wide variety of implementation and visualization variants of brushing and linking (Koytek et al. 2017), the highlighting feature focuses on a very basic visualization. It does not allow output-directed programming (since the code is not directly changed) but uses the same provenance information to find and highlight all changeable basic values, on which the selected waypoint depends. This means that for a selected waypoint, not the directly corresponding *moveTo* block but the values that determine its *x*, *y*, *z* position and rotation around the *z*-axis are highlighted (cf. Fig. 2a). As the highlighting mapping aid directly maps elements of the preview to code elements, it is an implementation of *element-to-element* mapping.

**Dynamic Linking** The waypoints shown in the preview are movable. The provenance of values that place the waypoint is tracked in the DSL during the live evaluation. Using this



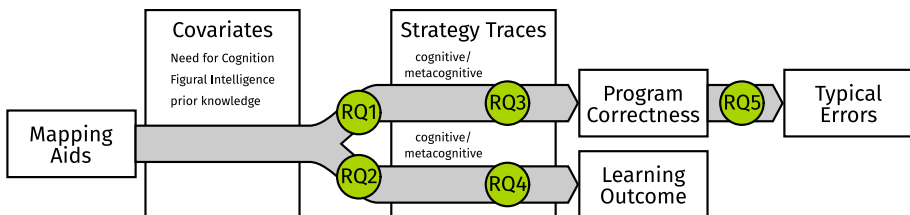
**Fig. 2** Realization of mapping aids in the PLE

information, an inverse to the code evaluation is calculated, and the changed position (due to the waypoint being moved) is applied directly to the code. For example, changing the position of the preview using the green arrow in Fig. 2b along the  $y$ -Axis uses the saved location information associated with the value ( $-1$  in the example) and tracks it back to its origin  $L2$  (shown in red). After the preview is re-rendered, the change to the new value 2 is visible, giving immediate feedback to the user (Hempel et al. 2019; Witte and Tichy 2019). Of course, the relation between the value at the location and the changed value is often more complex, e.g., if the changed value is calculated from the origin value. In this case, the calculation is first reversed using the provenance information of the value, before the target value is applied. This dynamic linking mapping aid is a form of *relation-to-relation* mapping, as more comprehensive structures and their relation between representations, as well as the insight into underlying processes are given (Gentner 1983; Seufert 2019).

As this linking between code and preview is performed at the fundamental level of the programming language, no semantic knowledge of the *moveTo* block or the waypoints in the preview is needed. Any (intermediate) result of a computation in the user's program can be automatically tracked back to the literal values used to compute it by analyzing and following its recorded history.

### 3 Research Hypotheses

This study and its methodology are based not only on the problem statement from the perspective of software engineering but also on findings from the research field of learning and instruction, as well as important cognitive psychology findings. In the following, we present our more specific hypotheses based on our research questions (RQs; Fig. 3). Our RQ1 and RQ2 look at the effect of the mapping aids to *program correctness* and *learning outcome*; *traces of cognitive and metacognitive strategies* in this process are considered in RQ3 and RQ4. Furthermore, RQ5 investigates *typical errors* in the different experimental groups. Based on the literature, we proposed more specific hypotheses (H) for each research question.



**Fig. 3** Flow chart displaying the relevant factors and related research questions with independent variables (mapping aids), the co-variables influencing the learning of the novices programmers followed by different dependent variables (learning strategy traces, program correctness, learning outcome and the typical error) that are influenced by the setting and the individual co-variables of the learners

**RQ 1: How effective are provenance-based mapping aids at increasing (semantic) program correctness?**

In particular, we expect...

- H1a a beneficial effect of highlighting on program correctness of the quadcopter missions
- H1b a beneficial effect of dynamic linking on program correctness of the three quadcopter missions
- H1c a synergetic effect of both mapping aids compared to a single mapping aid on the program correctness of the different missions

Based on previous findings, we expected that by highlighting, the visual attention of users is guided and they can see the corresponding elements. Hence, unnecessary visual search is reduced and the selection processes are eased as the respective elements do not need further consideration to determine whether they are corresponding or not (Ozcelik et al. 2010; Fries et al. 2021). Relation-to-relation mapping aids (dynamic linking) support the users as they outline the underlying processes by showing one possible solution for the user-induced change in the preview by automatically altering the respective coordinates in the code (Seufert 2019; Mayer et al. 2018). As both highlighting and dynamic linking might ease the programming process, both mapping aids in combination are expected to show a synergetic effect (Rey 2011).

**RQ 2: How effective are provenance-based mapping aids at improving learning outcomes?**

In particular, we expect...

- H2a the strongest beneficial effect of highlighting on the comprehension and application level of learning outcome (compared to the knowledge level of learning outcome)
- H2b the strongest beneficial effect of dynamic linking on the comprehension and application level of learning outcome (compared to the knowledge level of learning outcome)
- H2c a synergetic effect of both mapping aids compared to a single mapping aid on the different levels of learning outcome

Previous studies showed beneficial effects of mapping aids compared to control groups without additional help connecting the different representations in the PLE (Rey 2011; Patwardhan and Murthy 2017). Based on theoretical considerations, the mapping aids chosen trigger specific cognitive learning processes. Deeper understanding and learning are achieved, improving comprehension and application. Element-to-element mapping (highlighting) is assumed to simplify the finding of corresponding elements and, by this, the integration of the different representations (Fries et al. 2021). Furthermore, our relation-to-relation mapping aid (dynamic linking) outlines the connection of different components in the PLE and the consequences of manipulating one component (the live preview) onto another component (the code editor). This effect is expected to reflect mainly on higher



levels of learning outcomes (Patwardhan and Murthy 2017). Based on the theories presented and previous research, it is plausible that users would benefit more from the combination of both mapping aids than from one mapping aid at the different levels of learning outcome (Patwardhan and Murthy 2017; Rey 2011).

#### RQ 3: How are learning strategy traces and program correctness related?

We hypothesized a positive correlation of program correctness and ...

- H3a cognitive organizing strategies traces (changes in the preview)
- H3b cognitive elaboration strategies traces (time engaging in the task)
- H3c metacognitive planning strategies traces (time before starting with the task)
- H3d metacognitive monitoring strategies traces (number of started simulations)

As mentioned above, traces of cognitive and meta-cognitive learning strategies such as organizing and elaborating can be inferred from process data (Hadwin 2021; Jeske et al. 2014; Wild and Schiefele 1994). Based on the literature, it is expected that purposeful interaction in the PLE using strategies will lead to better performance in terms of program correctness (de Jong and van Joolingen 1998; Patwardhan and Murthy 2017).

#### RQ 4: How are learning strategy traces and learning outcome related?

We hypothesized a positive correlation of overall learning outcome and ...

- H4a cognitive organizing strategies traces (changes in the preview)
- H4b cognitive elaboration strategies traces (time engaging in the task)
- H4c metacognitive planning strategies traces (time before starting with the task)
- H4d metacognitive monitoring strategies traces (number of started simulations)

In line with the third research question that a positive effect of the strategy traces on the program correctness is expected, a positive effect of these strategy traces on overall learning success is also expected (Hadwin 2021; Wild and Schiefele 1994; Patwardhan and Murthy 2017).

#### RQ 5: What are typical errors and how do these mapping aids help novices reduce these errors?

Based on these previous findings, we expected...

- H5 the odds of making typical programming errors in the first mission to differ between experimental groups

Our expectation was based on previous publications that analyzed the difficulties of programming novices, and we explored the data to find a common or typical error pattern in the first mission (Prather et al. 2018; Patwardhan and Murthy 2017). We used data from the first (easiest) mission to ensure a high enough number of correct solutions across all experimental groups.

## 4 Method

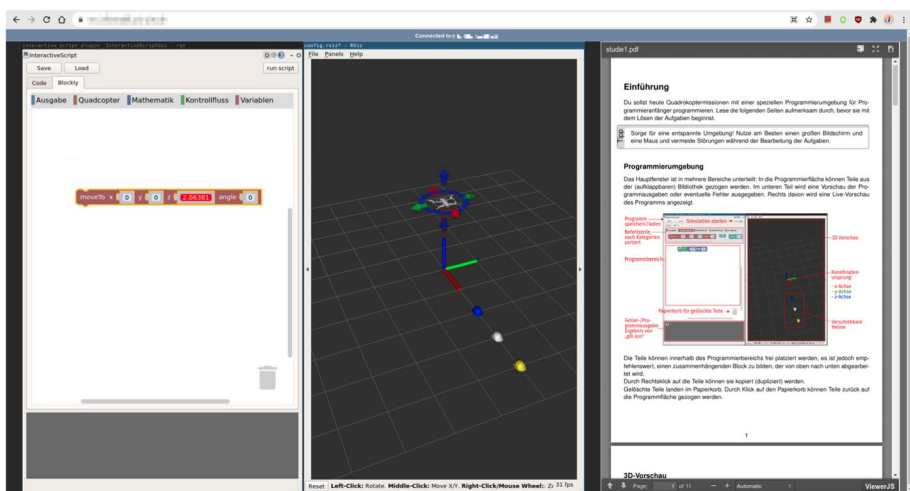
In the following section, we provide more details about the PLE including the programming task, the procedure (cf. Fig. 6), participants, study design, and the used questionnaires as well as the data preparation.

### 4.1 Setup of the Study PLE and its Mapping Aids

We developed an interactive online PLE by adapting the *interactive script* editor described in Witte and Tichy (2019) and previously introduced in Section 2.5. To fit the requirements of the study, the existing editor was modified to 1) include instructions to guide the participants through the different exercises (cf. Appendix B), 2) be accessible online and without installation through a web browser, and 3) implement the different experimental groups (i.e., enable or disable the mapping aids) and collect process data (screenshots, save states) to document participants' solution progress.

The PLE includes different views to display the code for the virtual quadcopter using a block-based editor, and a 3D preview of the planned flight trajectory. In addition, instructions for the programming exercises are displayed alongside the environment (Fig. 4).

On the left side of the PLE, the Blockly-based visual code editor (Pasternak et al. 2017) for a domain-specific language to define quadcopter missions was shown. This presentation format allowed the users to create their code through graphical manipulation as it represented program primitives, e.g., values, statements, and expressions, as interlocking blocks. New program blocks can be dragged from a library into the program canvas. Additionally, the program output was shown in a live console below, allowing users to print and inspect values during development, as well as during simulation execution. In the center, an interactive 3D preview of the planned mission of the quadcopter was displayed. Users were able to choose their perspective for the Cartesian coordinate system. To facilitate orientation in this preview, a grid on the ground plane, the origin and colored indicators along the x,y, and



**Fig. 4** Online PLE as seen by a participant of the study: code editor (left), 3D preview (center), and instruction PDF (right)

z-axis were shown. Three colored objects that could be dragged in the 3D view could be used as reference points, and their pose could be queried in the code. The quadcopter pose at each waypoint in the code was displayed in this preview, with lines connecting subsequent waypoints. By pressing the button "run script", a simulated quadcopter could be observed, while flying along the planned trajectory.

In the control group, only the 3D preview of the planned flight trajectory without any mapping aids was available. Users in the other groups could use one or both of the aforementioned mapping aids (cf. Section 2.5), resulting in four total groups: *no mapping aids*, *only highlighting*, *only dynamic linking*, and *both mapping aids*.

The study was conducted remotely on the participants' personal computers. To avoid complicated installation procedures, the PLE was executed on a server with a browser-based remote connection tool that displays the application in the browser window and sends inputs to the server. Inevitably, this introduces the risk of connection losses, as well as latency issues causing delayed reaction to input. In our testing and through questionnaire items asking for any technical issues during the study, we found that these issues affecting only a few participants. If the connection was lost, the page could simply be reloaded with no progress lost as the application still ran on the server. Furthermore, running the application on the server helps controlling the PLE: the application had a fixed resolution that was scaled to the participant's window size, ensuring that each participant had a similar viewport and performance. Screenshots of the PLE were automatically made without the risk of capturing private data on the participant's personal computer or requiring the installation of potentially intrusive software on the client computer to do so.

On the server, the full ROS<sup>2</sup> environment, needed to run *rViz*, *rqt* (including the *interactive\_script* plugin (Witte and Tichy 2019)), and the quadcopter simulation, was executed in a docker container<sup>3</sup> accepting remote connections through *vnc*. In order to reduce the complexity of the server infrastructure, only a single instance without scaling for supporting multiple concurrent users was used; therefore only one participant at a time participated in the study.

## 4.2 Programming Tasks – Quadcopter Missions

In the accompanying instruction PDF, users were asked to implement three different quadcopter missions of increasing complexity, which involved planning three flight trajectories (see Appendix B).

Consistent with our notion of novice (Section 2.1), these missions might seem trivial to any experienced programmer but introduce several concepts like loops, variables, and values calculated at run time. Due to the short time frame to solve the exercises, the developed programs are very short and lack more advanced concepts to structure code such as functions, complex data structures, or complex control flow. However, these concepts would require more experienced participants or multiple training sessions.

<sup>2</sup>Robot Operating System, <https://ros.org/>.

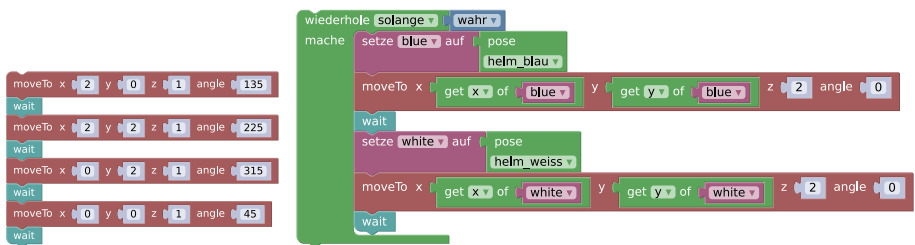
<sup>3</sup><https://www.docker.com/>. The PLE container is available on our github page at [https://github.com/sp-uulm/interactive\\_script/releases/tag/StudyPLE](https://github.com/sp-uulm/interactive_script/releases/tag/StudyPLE).

### Quadcopter Missions (M)

- M 1 Simple square: placing four waypoints in a square shape with the quadcopter facing towards the center in each.
- M 2 Commute between objects: the quadcopter flies between two waypoints dynamically defined by the position of two movable objects.
- M 3 Dynamic square: Combining elements of the first two missions by calculating the square's size depending on the position of the movable objects.

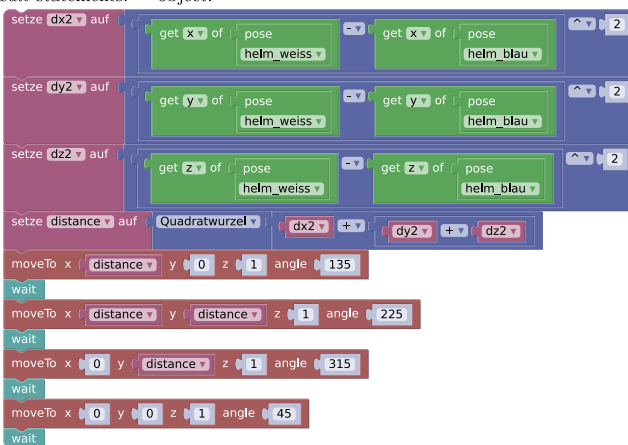
The solutions for each quadcopter mission is shown in Fig. 5.

To ensure that the learners are able to use the PLE, as well as to ensure that they understand the instruction, we consulted two experts and had three test participants for a pilot run. Based on these findings, we removed some smaller bugs in the PLE and added more details to the description. In addition, we added some hints to reduce the initial programming barrier. This was done by adding eight additional hints about programming in general. For instance, users were advised to divide one challenging task into easier sub-tasks to solve the missions. Qian and Lehman (2017) identify three major clusters of problems novice users face: to help participants focus on the *semantic understanding* of the language and environment, the *syntactic complexity* is reduced by using a block-based language and the



(a) Mission 1: A simple sequence of *moveTo* and *wait* statements.

(b) Mission 2: Participants needed to create a loop and get the position of an object.



(c) Mission 3: Participants needed variables and more complex expressions.

**Fig. 5** Possible solutions for each of the 3 exercises, that the participants had to implement. Images from Witte (2024)

difficulties in *problem solving* are reduced by providing hints and steps to solve the problem for each mission.

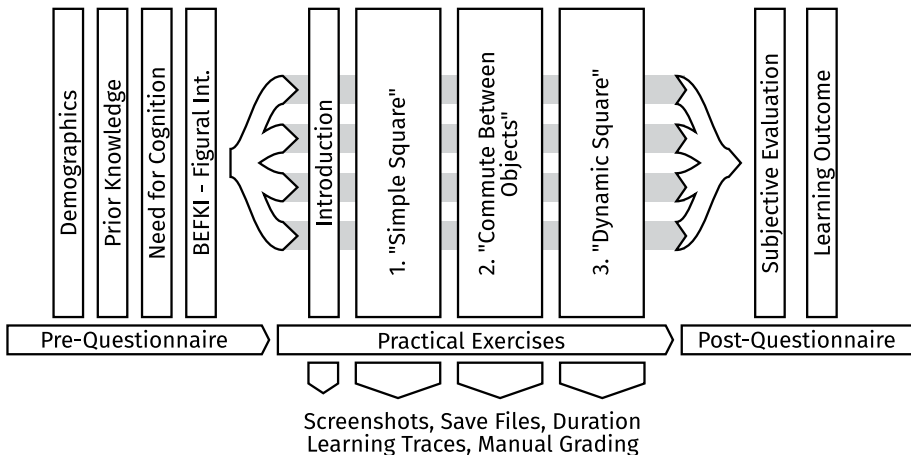
Furthermore, a short explanation of the different components of the interactive PLE was included. Users were informed about the mapping aids available depending on their experimental conditions. Before starting the first mission, users had to simply recreate a given example (see Fig. 6). This example enabled users to get to know the interactive PLE and asked them to try out the mapping aids if available.

### 4.3 Procedure

At the beginning of the online study, an overview of the study, as well as informed consent was presented to the participant using an online pre-questionnaire (see Fig. 6). The participants were aware that they could withdraw the study and the related data at any point in the study without any disadvantage. The pre-questionnaire started with demographic questions, and the pre-test included the prior knowledge questions. At the end of the pre-questionnaire, a link was included to log into the interactive PLE to program the three quadcopter missions. After finishing the three programming tasks, users completed a post-questionnaire that included the post-test learning outcome questionnaire and the opportunity to give feedback on the provided mapping aids (if these were available). At the end of the study, they were able to comment on the study and report technical problems during the study. In a separate online survey, they were able to leave their personal information to receive their compensation (a voucher). In total, the study took about 1.5 hours.

### 4.4 Participants and Study Design

The sample needed to investigate the effects of the chosen mapping aids was estimated based on an a priori power analysis using G\*Power version 3.1.9.4 (Faul et al. 2009). The potential effect size of the chosen mapping aids was calculated based on the results reported by van der Meij and de Jong (2006) with  $f^2(V) = .10$  (medium effect size, Cohen, 2013);



**Fig. 6** Elements of the study: questionnaires and programming exercises

$\alpha = .05$ ; power  $(1 - \beta) = 0.9$ . By this, we determined the minimum number of participants required with  $N = 91$ . The main study was conducted after a pilot phase that included feedback from 2 participants. Initially, 93 participants, who were mainly university students in psychology, participated in our online study. Because of technical problems (connection problems to the PLE, data recording), we excluded eleven participants from the further analysis. The remaining  $N = 82$  participants (26.50% male) were between 19 and 39 years old. ( $M_{age} = 25.02$ ;  $SD_{age} = 4.04$ ). We applied a 2x2 between-subject design randomly assigning the participants to one of the four different design options: with highlights and dynamic linking ( $n = 20$ ), with dynamic linking ( $n = 21$ ), with highlights ( $n = 22$ ) and the control group ( $n = 19$ ). As a dependent variable, the program correctness of three quadcopter mission tasks was assessed. Additionally, the learning outcome was measured on three different levels: knowledge, comprehension, and application. While participants were working on the programming tasks, screenshots of the PLE were automatically taken every five seconds to track progress and gain insight into the problem solving process. The program code and layout were automatically saved with a timestamp after each change. In addition, participants could create manual save files and were instructed to do so after every exercise. This allowed further insights into strategies and a typical error. The study was part of a larger research project and this paper focuses on the aforementioned research questions. Figure 6 gives an overview of the sequence of the study, the contents of the pre-and post-questionnaires, and the measured variables.

## 4.5 Questionnaires

In an online pre-questionnaire, participants were asked about their gender, age, educational level, field of study, and any prior experiences with programming and specific programming languages. To gain further insights into their domain-specific prior knowledge, we developed eight questions about basic concepts of programming and relevant concepts for the programming missions. A pre-test included four knowledge questions to measure domain-specific prior knowledge: In addition to three basic definitions of the terms GUI, debugging, and logical operators, users also had to name a waypoint in the coordinate system. Another four questions were created (e.g. *'Name one major difference between dynamically and statically typed programming languages.'*) to capture a more in-depth understanding of programming concepts. To ensure good quality of measurement of prior knowledge, we analyzed the inter-rater reliability, which revealed very high consistency between the two ratings of the independent raters ( $r = .993$ ,  $p < .001$ ,  $CI_{95\%} = .989 - .995$ ).

To measure users' need for cognition, we used the German version of the short scale for the need for cognition (Beißert et al. 2014). The users had to rate four items (e.g., *'I like my life to be full of tricky tasks that I have to solve.'*) on a seven-point Likert-scale. In line with the published retest-reliability of  $r = .78$ , the scale was assumed to be reliable (Beißert et al. 2014). In our sample, we determined a McDonald's omega of  $\omega = .731$  ( $CI_{95\%} = .640 - .823$ ), which was acceptable.

We included a measurement for fluid intelligence, as both spatial ability and general logical ability are relevant to interacting with the PLE, as well as learning the relevant concepts. We used the figural module of BEFKI 11+ that includes 16 items (Schipolowski et al. 2017). Dimitrov's scale reliability for the published sample is  $\rho = .81$  (total scale  $gf$  where the figural model is a part in; Schipolowski et al. 2017).

We developed a post-test to measure learning outcomes, which consisted of nine questions on the three levels (knowledge, comprehension, application, Bloom et al. 1956). To measure the learning outcome on the knowledge level, three multiple choice questions were developed (e.g., *‘Which statements about variables are correct? a) Only numbers and words can be assigned as values. b) A variable can be assigned multiple times. c) A variable can always be used instead of the value assigned to it. d) The value of a variable is unchangeable.’*). For the comprehension level, two open questions were included (e.g., *‘Name a basic difference between the pose block and the wait block.’*). The second comprehension question included three short programming sequences and their effects on the flying trajectory should be described. For application, we developed four questions. Two questions referred to finding of bugs in a given code. In the other two questions, users needed to map the corresponding code and preview as well as describe the change in the code based on two different simulation results in the preview. The questions of the post-test aimed to examine different aspects of the learning content. Different questions referred to different concepts and processes. Hence, we expected no high internal consistency of the different questions. To ensure that the learning outcome was measured in a rigorous way, we analyzed the inter-rater reliability, which revealed a very high consistency between the two ratings ( $r = .999$ ,  $p < .001$ ,  $CI_{95\%} = .998 - .999$ ).

As the study was part of a larger research project, the cognitive load of novices was measured and a subjective assessment of it was made, but is not further reported in this paper (see Appendix A).

#### 4.6 Learning Process Data and Program Correctness

We analyzed traces that might hint to certain underlying (cognitive or meta-cognitive) learning strategies to gain deeper insights into the approaches novices used to program the quadcopter mission, based on process data (Hadwin et al. 2007; Patwardhan and Murthy 2017; Wild and Schiefele 1994). In line with previous publications (e.g. Matcha et al. 2019; Fincham et al. 2018), we defined certain indicators for traces of different learning strategies.

As cognitive learning strategies, organizing and elaboration were considered. For organizing, it is assumed that novices worked with the preview to comprehend the structure of the code without changing the code. For example, perspective changes were recorded here, or scaling of the preview (zoom-in(-out)). To capture this, a threshold of 30 seconds was set in which no code changes took place, but the preview had to be changed. To assess elaboration in the PLE, the time during which novices were (actively) engaged in the task was considered.

For meta-cognitive learning strategies, planning and monitoring were considered. The planning phase was measured as the time (after completion of a task) until the start of the (next) task. For monitoring, the number of simulation runs in the preview was chosen as the metric.

Based on the manual save files and the screenshots at the end of each quadcopter mission, each novice’s solution was graded. The solution of each task was critically examined by two experts on the basis of a specific grading scheme (Appendix C). This allowed for the objective assignment of points for the solution (program correctness) of each mission. In addition, the solution scheme allowed for the identification of typical errors in the tasks. An exploratory approach was used to collect similar errors in the solutions of different



participants in all experimental groups. The identified typical errors were then located in the process data (screenshots) of all participants.

## 4.7 Data Preparation

The online questionnaires were presented using the Unipark tool. We prepared and analyzed the data using R 4.1.0. Process data were analyzed both automatically and by using a specific labeling tool developed for this use case: consecutive unchanged screenshots were automatically filtered out to reduce redundant data. The timestamps of automatic saves and screenshots were then used to automatically classify user activity (e.g., changing the program, using the preview) and calculate metrics, such as time and duration of user activity. Then, the start and end of programming exercises, as well as usage of the mapping aids and simulation runs, were manually labeled using a custom labeling tool on screenshots and saved programs. A weighted mean of the prior knowledge test and the learning outcome scores were calculated for each user, which was based on the rating of two raters. For figural intelligence and need for cognition, the scores were calculated using the published solution schemes (Beißert et al. 2014; Schipolowski et al. 2017). Furthermore, we checked for outliers and whether assumptions for parametric testing were met (normal distribution, homogeneity of variances, and covariances). To ensure high testing power for the analysis of our hypothesis and to avoid accumulation of alpha error, we used MANOVAs or MANCOVAs. In addition, we used contrasts to analyze our hypotheses. Due to this, we were not relying on less powerful post-hoc testing (Field 2013).

Table 1 summarizes all the data collected during the study and their usage to answer our research questions.

## 5 Results

### 5.1 Descriptive Results and Assumption Testing

The domain-specific prior knowledge of the learners was medium, in accordance with the idea that our sample contains novices in programming (see Table 2). For figural intelligence, we found medium values in all four experimental conditions (see Table 2). To ensure that there were no systematic differences between experimental groups, we conducted a

**Table 1** Overview of collected study data, scales and questionnaires used and its usage

Data Collected	Scale/Details	Function
<i>Pre-Questionnaire</i>		
Prior knowledge	Appendix D.1	} Covariates (RQ1 – 4)
Figural knowledge	Schipolowski et al. (2017)	
Need for cognition	Beißert et al. (2014)	
<i>Practical Exercises (Mission 1–3)</i>		
Manual save files	} Appendix C, Section 4.7	} Program Correctness (RQ1, 3), Strategy Traces (RQ3, 4), Typical Errors (RQ5)
Automatic save files		
Screenshots		
<i>Post-Questionnaire</i>		
Learning test	Appendix D.3	Learning Outcome (RQ2,4)

**Table 2** Means and standard deviations of the experimental groups on different variables without considering the covariates

	No support	Highlights	Dynamic linking	Both mapping aids
	$n = 19$	$n = 22$	$n = 21$	$n = 20$
	M (SD)	M (SD)	M (SD)	M (SD)
<i>Novices' characteristics in %</i>				
Prior knowledge	43.37 (19.92)	37.50 (19.92)	38.19 (18.60)	45.99 (20.78)
Figural knowledge	55.40 (17.50)	57.20 (18.10)	58.80 (17.20)	54.80 (10.80)
Need for cognition	66.35 (17.91)	61.53 (19.28)	64.29 (16.56)	66.43 (15.26)
<i>Dependent Variables in %</i>				
Program correctness				
Mission 1	76.32 (19.50)	79.55 (17.01)	72.22 (15.21)	90.00 (13.68)
Mission 2	67.54 (33.09)	64.39 (24.83)	62.70 (23.51)	85.00 (20.16)
Mission 3	26.75 (39.14)	27.27 (41.96)	33.33 (39.44)	54.17 (46.48)
Learning outcome				
Knowledge	66.23 (18.94)	62.88 (22.96)	67.46 (20.40)	71.67 (23.32)
Comprehension	29.39 (33.55)	32.67 (23.77)	41.27 (27.40)	46.98 (25.55)
Application	49.67 (31.63)	42.61 (33.33)	51.79 (26.01)	66.88 (26.99)

MANOVA that included prior knowledge, figural intelligence and need for cognition. Taking into account learners' preconditions, no significant differences could be found between the experimental groups ( $F < 1$ ,  $p > .454$ ). Furthermore, a multivariate normal distribution was tested, but not found for all critical variables for each experimental group. Conducting parametric MANCOVA is still assumed to be robust and favorable compared to non-parametric testing based on the recommendations of Finch (2005). The homogeneity of the variance was supported by the Levene-Test ( $p > .062$ ). The homogeneity of the covariances was checked using the Box-Test, based on which homogeneity was assumed ( $p > .868$ ). As both program correctness and learning outcome were claimed to be performance measures, a positive relationship between these two concepts was expected. In line with this, we found a strong positive correlation  $r = .71$  ( $p < .001^{***}$ ) between program correctness and learning outcome.

## 5.2 Inferential Testing

We conducted a MANCOVA including the different experimental groups represented by the two factors highlighting and dynamic linking to analyze our hypotheses concerning the main effect of the mapping aids. We chose prior knowledge, figural intelligence, and need for cognition as covariates. As dependent variables, we included the program correctness of

the three quadcopter missions (RQ1) and all three levels of learning outcome (knowledge, comprehension, application; RQ2). To test our hypotheses, we included simple contrasts for each mapping aid (highlights, dynamic linking). For analyzing the synergetic effect, we additionally included contrasts comparing the three experimental conditions with mapping aids. To analyze our hypothesis concerning the relationship between learning strategy traces and performance, a bivariate correlation was utilized (RQ3 & RQ4). In addition, we looked into the collected process data. We calculated a logistic regression to gain insight into the probability of an exemplary typical error (RQ5).

### 5.3 Effects on Program Correctness (RQ1)

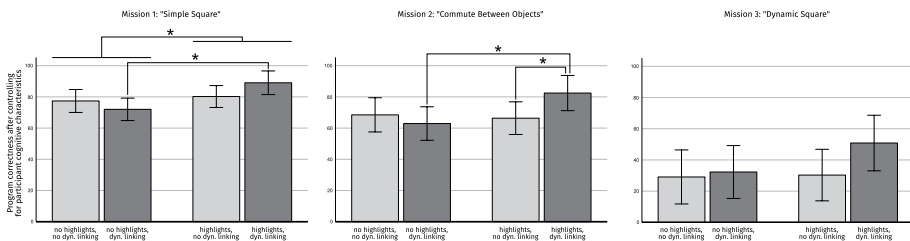
As both mapping aids showed interrelations between the different parts of the interactive PLE, we expected that both mapping aids would have a beneficial effect. As mentioned, we used the results of the conducted MANCOVA including the contrasts for each mapping aid and the program correctness of all three quadcopter missions, to analyze our hypotheses.

We expected a beneficial effect of highlighting on program correctness (H1a). Based on the descriptive means of program correctness in all three tasks, the means were higher in the experimental groups with highlighting in missions 1 and 2 (represented by the bars of the group with highlighting and both mapping aids in Fig. 7).

Based on the MANCOVA, a significant beneficial effect was only found in the first mission (mission 1:  $p = .008^*$ ,  $\eta^2_{\text{partial}} = .091$ ) but not for the following missions (mission 2:  $p = .112$ ,  $\eta^2_{\text{partial}} = .033$ ; mission 3:  $p = .252$ ,  $\eta^2_{\text{partial}} = .017$ ). Thus, our hypothesis was only partially supported by the data (H1a).

Additionally, we expected a beneficial effect of dynamic linking on correctly solving the programming tasks (H1b). Based on descriptive means, no substantial difference was found in the first task between the groups with and without dynamic linking. Descriptively, higher means were given in the groups with dynamic linking (shown by the bars of dynamic linking and both mapping aids in Fig. 7) compared to the groups without dynamic linking in the second and third tasks.

We found no significant beneficial effect for dynamic linking on solving the three tasks (mission 1:  $p = .636$ ,  $\eta^2_{\text{partial}} < .003$ ; mission 2:  $p = .355$ ,  $\eta^2_{\text{partial}} = .012$ ; mission 3:  $p = .171$ ,  $\eta^2_{\text{partial}} = .025$ ). Based on these findings, our hypothesis was not supported (H1b).



**Fig. 7** Means and standard error ( $CI = 95\%$ ) of the program correctness (in %) of the three quadcopter missions depending on the experimental group

When both mapping aids were combined, the descriptive values suggest that this made quadcopter mission programming easier compared to using just one mapping aid (synergetic effect, H1c). In addition to the reported results of the MANCOVA (see Table 3), we analyzed the differences between the groups with mapping aids with respective contrasts, to gain insight into whether the combined (synergetic) mapping aid (dynamic linking & highlighting) results in a significantly higher performance compared to single mapping aids (dynamic linking or highlighting). We found a significant beneficial synergetic effect for both mapping aids compared to dynamic linking for the first and second mission ( $p_{\text{mission1}} = .002^*$ ,  $p_{\text{mission2}} = .015^*$ ) but not for the third mission ( $p_{\text{mission3}} = .135$ ). Furthermore, we found a beneficial synergetic effect for both mapping aids compared to highlighting for the second mission ( $p_{\text{mission2}} = .042^*$ ) but not for the first or third mission ( $p_{\text{mission1}} = .095$ ,  $p_{\text{mission3}} = .099$ ).

Hence, hypothesis (H1c) was partially supported by the data.

## 5.4 Effects of Highlighting and Dynamic Linking on Learning Outcomes (RQ2)

We expected to find stronger beneficial effects of highlighting on the comprehension and on the application level of learning outcome compared to the knowledge level (H2a). Descriptively, we found higher means for the comprehension and the application level in the experimental group with highlighting compared to the control group, while the knowledge level merely differed (see Fig. 8). Based on our MANCOVA, we found no significant main effect of highlighting on the three levels of learning outcome ( $p > .265$ , see Table 3). Thus, our hypothesis was not supported by the data (H2a).

In our next hypothesis, we expected the strongest beneficial effect of dynamic linking on the comprehension and on the application level (H2b). Descriptively, in the groups with dynamic linking, the means of all levels of learning outcome were higher compared to the groups without dynamic linking. We found a significant main effect of dynamic linking on the comprehension level of learning outcome  $F(1, 75) = 5.61$ ,  $p = .020^*$ ) with a medium effect size ( $\eta^2_{\text{partial}} = .070$ ) and a significant effect on application ( $F(1, 75) = 4.08$ ,  $p = .047^*$ ,  $\eta^2_{\text{partial}} = .052$ ): There were no significant beneficial effects of dynamic linking on the knowledge level ( $p = .450$ ,  $\eta^2_{\text{partial}} = .008$ ). Hence, our hypothesis was supported by the given data, as the strongest effect of dynamic linking was found for the comprehension and application level (H2b).

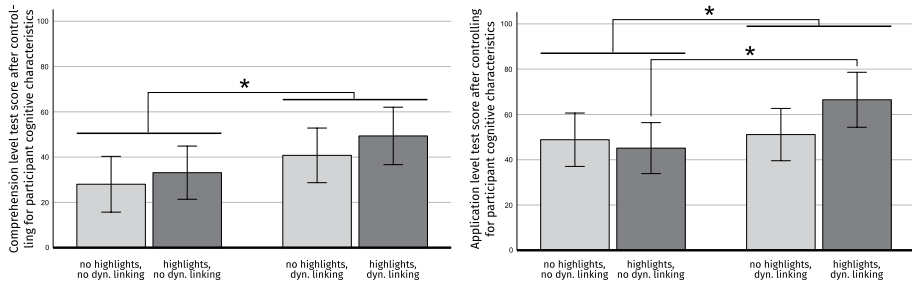
When comparing the different mapping aids for their beneficial effect on different learning outcomes (H2c), the descriptive pattern implied stronger beneficial effects of both mapping aids compared to a single one (see Fig. 8). For testing hypothesis H1c, we used a contrast analysis to complement the MANCOVA results and to allow insight into the specific differences between the groups with mapping aids.

In our hypothesis, we assumed a synergetic effect of both mapping aids compared to a single mapping aid on the different levels of learning outcome (H2c). Based on our MANCOVA contrast analysis, we found a significant beneficial synergetic effect for both mapping aids compared to highlighting on the application level ( $p_{\text{application}} = .013$ ), only ( $p_{\text{knowledge}} = .317$ ,  $p_{\text{comprehension}} = .068$ , see Fig. 8).

**Table 3** Results of the between-subject effects of the MANCOVA with the different levels of learning outcome, program correctness, simple contrasts for element-to-element mapping aid (highlights), and relation-to-relation mapping aid (dynamic linking), with learners' figural intelligence, prior knowledge and need for cognition as covariates measured in the pretest to control for participants cognitive characteristics

		$F(1, 75)$	$p$	$\eta^2_{\text{partial}}$
<i>Knowledge</i>	Highlights	0.01	.940	< .001
	Dynamic linking	0.58	.450	.008
	Dynamic linking * highlights	0.44	.507	.006
	Prior knowledge	0.93	.338	.012
	Figural intelligence	49.0	.486	.006
<i>Comprehension</i>	Need for cognition	0.03	.872	< .001
	Highlights	1.26	.265	.017
	Dynamic linking	5.61	<b>.020</b>	.070
	Dynamic linking * highlights	0.08	.782	.001
	Prior knowledge	0.10	.752	.001
<i>Application</i>	Figural intelligence	0.36	.552	.005
	Need for cognition	1.19	.279	.016
	Highlights	1.00	.321	.013
	Dynamic linking	4.08	<b>.047</b>	.052
	Dynamic linking * highlights	2.55	.114	.033
<i>Program correctness mission 1</i>	Prior knowledge	0.97	.327	.013
	Figural intelligence	4.21	<b>.044</b>	.053
	Need for cognition	11.5	<b>.001</b>	.133
	Highlights	7.47	<b>.008</b>	.091
	Dynamic linking	0.23	.636	.003
<i>Program correctness mission 2</i>	Dynamic linking * highlights	3.64	.600	.046
	Prior knowledge	0.00	.994	< .001
	Figural intelligence	0.44	.511	.006
	Need for cognition	3.72	.058	.047
	Highlights	2.59	.112	.033
<i>Program correctness mission 3</i>	Dynamic linking	0.94	.335	.012
	Dynamic linking * highlights	3.83	.054	.049
	Prior knowledge	1.13	.291	.015
	Figural intelligence	0.41	.525	.005
	Need for cognition	6.93	<b>.010</b>	.085
<i>Program correctness mission 3</i>	Highlights	1.34	.252	.017
	Dynamic linking	1.91	.171	.025
	Dynamic linking * highlights	0.99	.324	.013
	Prior knowledge	2.70	.104	.035
	Figural intelligence	6.34	<b>.014</b>	.078
<i>Program correctness mission 3</i>	Need for cognition	3.37	.070	.043

A comparison of both mapping aids and dynamic linking did not reveal significant differences ( $p_{\text{knowledge}} = .678$ ,  $p_{\text{comprehension}} = .335$ ,  $p_{\text{application}} = .074$ ). Hence, our hypothesis (H2c) was only partially supported by the data.



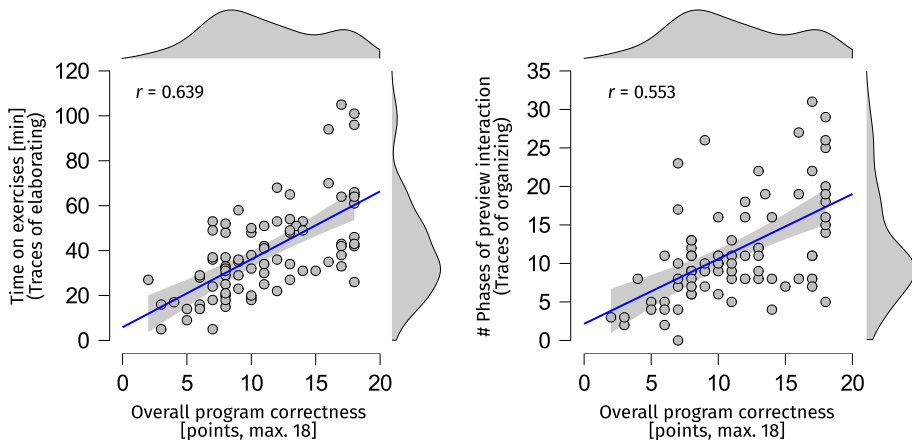
**Fig. 8** Effects of mapping aids on comprehension (left) and application level (right) of learning outcome measured after finishing the quadcopter missions in the PLE under consideration of the covariates displaying means and standard errors ( $CI = 95\%$ )

### 5.5 Correlation of Learning Strategy Traces & Program Correctness (RQ3)

Having a closer look at the learning behavior while programming the quadcopter mission, traces of cognitive (organizing, elaborating) and meta-cognitive (planning, monitoring) learning strategies were analyzed. We expected the traces to be positively correlated with the overall program correctness.

First, we analyzed the traces of the cognitive learning strategies. Based on scatter plot inspection, we found the expected pattern of organizing (indicated by interacting only with the graphical preview) and program correctness. We found a medium, positive correlation ( $r = .553$ ,  $p < .001^{***}$ , see Fig. 9).

Thus, our hypothesis (H3a) was supported. Inspecting the descriptive pattern of elaborating (trace indicated by overall time in the PLE), a positive relation with program correctness was expected. In line with our hypothesis (H3b), we revealed a strong positive correlation between elaboration and program correctness ( $r = .639$   $p < .001^{***}$ ). Having a look at the traces of meta-cognitive learning strategies, based on visual inspection, a positive relationship between planning (indicated by the time before starting the tasks) and program



**Fig. 9** Scatter plot displaying the relationship between traces of cognitive learning strategies and overall program correctness

correctness was not supported by the data. In line with this, no significant correlation was found ( $r = -.163$ ,  $p = .214$ ) and the hypothesis (H3c) was not supported by the data. For monitoring (indicated by starting the simulation in the preview), we found the pattern fitting to our hypothesized relation with program correctness. However, we did not find a significant correlation between monitoring and program correctness ( $r = .234$ ,  $p = .056$ ; H3d).

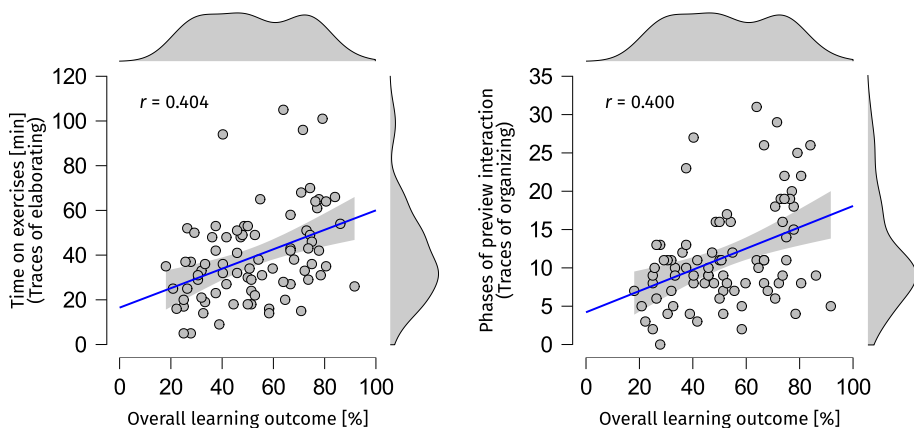
## 5.6 Relation of Learning Traces and Learning Outcome (RQ4)

We expected that the traces of cognitive and meta-cognitive learning strategies are positively related to overall learning outcomes. In line with the positive descriptive trend, we found a medium, positive correlation between traces of organizing and overall learning outcome ( $r = .400$ ,  $p < .001^{***}$ , see Fig. 10). Thus, our hypothesis (H4a) was supported.

Inspecting the descriptive pattern of elaborating, a positive relation with learning outcomes was expected. In line with this, the findings related to our hypothesis (H4b) revealed a medium positive correlation between elaboration and overall learning outcome ( $r = .404$ ,  $p < .001^{***}$ , see Fig. 10). Having a look at the traces of meta-cognitive learning strategies, again (see RQ3) a positive relationship between planning and program correctness was not supported by the data. In line with this, no significant correlation was found for monitoring ( $r_{\text{monitoring}} = .169$ ,  $p = .170$ ) nor planning ( $r_{\text{planning}} = -.013$ ,  $p = .924$ ). Thus, these hypotheses were not supported by our findings (H4c & H4d).

## 5.7 A Typical Error (RQ5)

In order to receive further insights into the novice's handling of the programming missions, we analyzed process data. This method provided further insight into the sources of a typical error and which mistakes were typically made. We focused our analysis on the first quadcopter mission to 1) reduce the very large data set to a manageable size, using a task that has a single correct solution without much variance and 2) avoid bias due to uneven representation of the different experimental groups. Although nearly all participants managed to create a reasonable solution for the first (easiest) mission, fewer participants without



**Fig. 10** Scatter plot displaying the relationship between traces of cognitive learning strategies and overall learning outcome



**Table 4** Coefficients of the logistic regression predicting the chance of correct angle in mission 1 considering the effect of experimental groups (1-3) against the control group (4) with figural intelligence, need for cognition, and prior knowledge as covariates

	<i>Estimate</i>	<i>SE</i>	<i>Odds Ratio</i>	<i>z</i>	<i>Wald<math>\chi^2</math></i>	Wald Test	
						<i>df</i>	<i>p</i>
(Intercept)	-2.387	1.427	0.092	-1.673	2.799	1	.094
Figural intelligence	0.418	1.509	1.519	0.277	0.077	1	.782
Need for cognition	0.013	0.015	1.013	0.859	0.738	1	.390
Prior knowledge	0.011	0.013	1.011	0.843	0.710	1	.399
Dyn. linking * highlights (1)	1.645	0.706	5.181	2.330	5.429	1	.020
Dynamic linking (2)	-0.077	0.704	0.926	-0.109	0.012	1	.913
Highlights (3)	0.537	0.675	1.711	0.796	0.634	1	.426

mapping aids were able to do so for the more difficult missions 2 and 3. We used the solution scheme (see Appendix C) to find examples of typical errors. To achieve a maximum score of 6 points, the following criteria had to be met: Creation of four different waypoints; these waypoints form a square; angles of the quadcopters were manipulated at least once; angles of the quadcopters were correctly adjusted as desired in the task; the quadcopter flies; each “moveTo” is followed by ‘wait’ or ‘sleep’.

In the first mission, setting the angle of the drones proved to be a typical source of error. Descriptively, it was found that about 57% ( $n = 47$ ) of the participants did not set the angle correctly and the distribution seemed to differ between the experimental groups.

Therefore, this typical error was analyzed in more detail as an example. We performed a logistic regression including covariates (see Table 4) to test whether the mapping aids offered had an effect on the performance in setting the correct angle. Based on our regression, participants with both mapping aids had a 5.18-times higher chance ( $p = .020^*$ ) of having set the angle correctly in the first task compared to the control group. Hence, our hypothesis (H5) was supported for this exemplary typical error. Further details on other typical errors are included in the Appendix E.

## 6 Discussion

The effect of mapping aids was considered based on a multi-method assessment approach. By this, we gained further insight into different aspects of potential benefits from the respective mapping aids. A first look at the descriptive data shows that a very differentiated approach is needed here to assess the positive effects of the individual support conditions and their combination: The support conditions provided did not help to improve the performance per se. In some cases, we see slightly decreased performance compared to the control group without help. This could be due to individual differences, additional load, or inferences as the support conditions offered could be in conflict with the existing learning strategy or solution patterns (e.g. Ruttun and Macredie 2012; Pavlova 2024).

### 6.1 Mapping Aids and Program Correctness (RQ1)

We expected a beneficial effect of the two mapping aids on program correctness. In contrast to our initial expectation, only highlighting increased performance for programming the first

mission, but no significant beneficial effects were found for dynamic linking. Additionally, the assumed synergistic effect was only found when comparing both mapping aids against dynamic linking (first and second missions) and highlighting (second mission).

Despite the fact that a positive effect of dynamic linking and combined mapping aids compared to single-help conditions is theoretically plausible based on cognitive processes, previous empirical studies describe similar findings as in our study (e.g. Rey 2011). With respect to the underlying cognitive processes, plausible reasons for the lack of a positive effect of dynamic linking can be described in addition to the specific application-related conclusions: An additional load on working memory due to the presentation and use of the aids could occur and result in lower performance (Lowe 1996; Sweller 2020). Reflecting on this, Rey (2011) concluded, based on theoretical inferences and heterogeneous empirical findings, that dynamic linking is not per se advantageous or disadvantageous, but that the specific implementation also plays an important role. Additionally, Seal et al. (2010) described that interactive elements could support users depending on the given task and emphasized that mapping aids might cause additional strains while learning. The three quadcopter missions had different levels of difficulty: While mission 1 only required the extension of the given code example, the subsequent missions contained further code elements and calculations that were not explicitly included in the example. The question now is whether certain aids or their combination are particularly advantageous for different partial solutions of the given missions. This question was examined in RQ5 based on the example of a typical error (see Section 6.4). In addition, the question arises as to what effect the mapping aids have at the learning level and whether, for example, they particularly support the comprehension and application of what has been learned. This was examined in more detail in the next research question.

## 6.2 Mapping Aids and Learning Outcome (RQ2)

Based on our hypotheses, we expected a beneficial effect of the two mapping aids on higher levels of learning outcomes. Beneficial effects were found for dynamic linking on comprehension and application level of learning outcome, but not for highlighting. A comparison of single aids with the combined condition revealed beneficial effects for adding dynamic linking at the application level.

Based on these findings, different conclusions can be drawn: (i) the knowledge level of learning outcome was not affected by the mapping aids or their combination. This was in line with our expectation as these mapping aids facilitated the integration of elements and processes in the PLE and not, in particular, the sub-semantic processing of the different components (Patwardhan and Murthy 2017; Fries et al. 2021). (ii) Beneficial effects on learning outcomes were found for dynamic linking for both comprehension and application while highlighting did not significantly increase the learning outcome. This was not in line with our initial expectations. However, a heterogeneous pattern for the effects of mapping aids can be described by a closer look at previous findings. In contrast to the advantages described, there are also studies (e.g. van der Meij and de Jong 2006) that do not describe the beneficial effects of mapping aids. Based on theoretical considerations, plausible explanations for these findings exist: When using highlights, novices are passively pointed to connections and no longer actively reflect on them, so deeper learning does not occur (Ainsworth 1999; Seufert and Brünken 2006). Providing highlights in combination

with a very specific task could, for example, encourage the use of means-end strategies (Erhel and Jamet 2019).

Identifying, e.g., two related elements, *including* their interaction, is particularly central to the level of understanding. This could be a possible explanation for why the highlighting was rather less helpful in answering the questions on the comprehension level, since no emphasis was placed on the underlying process, but only on the corresponding elements. In order to understand this finding more deeply, future studies should include further measurements for the development of mental models in order to better differentiate further sub-processes of information processing (for details see (Vogt 2021)). This will provide more precise information on the effects of highlights on cognitive processes.

For dynamic linking, novices received hints about connections between code and output. However, to maximize learning outcomes, they still needed to reflect on the relationships between the respective components, even though highlighting had already made these connections explicit (Gentner 1983). (iii) This is also reflected in the finding that sometimes the synergistic effect of both mapping aids on learning outcome was found, compared with the single aid conditions: novices were stimulated by adding dynamic linking to process the learning content more deeply, which was reflected in synergistic effects at the application level when comparing both mapping aids.

### 6.3 Traces of Learning Strategies and Performance (RQ3 & RQ4)

When analyzing the relationship between learning strategies and performance, it was found that only the traces of cognitive strategies were positively related to performance (program correctness and learning outcome). In contrast, these relationships with performance could not be found for the meta-cognitive strategies traces.

Reflecting critically on these findings, the following explanations can be found: Although the selected indicators or traces can be assigned to specific learning strategies, there are also other possible indicators (Roll and Winne 2015). Based on our findings, meta-cognitive strategy traces were not related to performance. In our study, planning was traced by the time before the start of programming each mission. However, it cannot be ruled out that novices also planned during the tasks. Hence, our findings reflect one general challenge in dealing with process data and deducing intentions or strategies based on behavioral traces (Malmberg et al. 2017).

In addition, the instructions for the missions were already transparently divided into sub-steps or sub-problems to support novices, making part of the structuring and planning process explicit. Consequently, it is plausible that planning in this specific case may not be strongly correlated with performance, as the task instructions already facilitated this aspect. Therefore, the operationalization of the meta-cognitive strategy trace, as well as the meta-cognitive strategies employed during the programming tasks, should be reconsidered in future studies.

In contrast, traces of monitoring showed a descriptive trend that indicated a positive correlation with program correctness, which was not significant. Again, the question is to what extent the monitoring already took place when the novices looked at the preview without running the simulation and whether further or better traces for the monitoring process can be found here, which would allow for even deeper and more differentiated insight into the

application of meta-cognitive learning strategies. Furthermore, a more sophisticated way to measure the elaboration during the task could be chosen.

## 6.4 Avoiding a Typical Error (RQ5)

In order to gain further insight into the solution of the programming missions, typical errors in mission 1 were analyzed. We focused our analysis on the first mission as the high rate of successful novices led to few but common mistakes that were well suited for detailed analysis. The lower success rate in, e.g., mission 3 leads to a high number of very diverse observed problems, as well as many incomplete solutions, that mask interesting, typical errors. The typical exemplary error was that novices did not set the correct angle of the quadcopter or did not adjust it at all. Through the screenshots, we observed participants using the dynamic linking feature to first set a rough angle and then correct it in the code. The analysis showed that novices with both aids had a significantly higher chance of setting the angle correctly.

Reflecting in detail on the role of the mapping aid in solving this partial problem, the findings are in line with expectations based on our theoretical assumptions. For instance, dynamic linking offers a direct manipulation of the angles. In addition, highlighting should be helpful for the orientation and assignment of the respective waypoints. When combining both mapping aids, novices should be supported in solving the angle setting in the first quadcopter mission. Hence, this exemplary partial problem was chosen to examine whether the odds of error changed depending on the available mapping aids. Typical problems for correct angle setting were that novices either did not change the angle at all or were not able to deduce the correct values for the required angles for each waypoint. In the condition with both mapping aids, they had the opportunity to adapt the 3D preview directly to set the angles correctly. Without dynamic linking, they had to calculate the angle. Without the highlights, they had to search for the respective waypoint without assistance to map the code and the waypoint in the preview.

In the future, guidelines could be developed based on available empirical studies, which would allow to deduce the type of mapping assistance in the context of the specific (sub) problem and, thus, enable targeted assistance in the individual (sub)tasks (Seal et al. 2010). Designers and providers of PLEs could benefit from such guidelines or hints to create appropriate support conditions for novice programmers depending on the tasks.

## 6.5 Investigating Aggregated Task Results

The previous sections covered the a-priori defined research design where our goal was investigating program correctness and learning outcomes for each task separately. In this section, we investigate both dependent variables again, now aggregating the results of all three tasks<sup>4</sup>.

The results of the MANCOVA revealed significant effects of the mapping aids ( $Df = 3$ ,  $F = 2.3339$ ,  $p = 0.03490$ ) on the dependent variables program correctness and learning outcomes. The covariate need for cognition was also significant ( $Df = 2$ ,  $F = 4.2678$ ,  $p = 0.01761$ ). The effect sizes (partial  $\eta^2 = 0.09$ ) for the mapping aids and need for cognition (partial  $\eta^2 = 0.10$ ) indicate a moderate effect size.

<sup>4</sup>This analysis was proposed by one of the reviewers.

Based on these results, we further investigated how the three treatments compared to the control group using Dunnett's test and found significant effects on program correctness between the group with both mapping aids and the control group ( $F = 7.69559$ ,  $p = 0.0195$ ) with a moderate effect size (partial  $\eta^2 = 0.09$ ). We did not find significant effects for learning outcomes ( $F = 5.3536$ ,  $p = 0.0629$ ).

Overall, these results indicate that having both mapping aids increases novices' programming performance. However, this analysis was not part of the initial research design. Hence, these results warrant further investigation and a follow-up study.

## 6.6 Limitations of the Study and Threats to Validity

The PLE was presented as an online tool in order to reach the necessary number of participants. This setting comes with some limitations. In a controlled laboratory study, the investigator could have ensured that the available aids were used (at least once). Despite the explicit hint to use the mapping aids in the initial example, not all subjects complied with this request. Furthermore, it could not be ensured that the novices did not use other (online or generative AI) support to answer the questions or to solve the programming tasks. To limit this problem, the tasks were very specific, used a custom DSL, and simple copying of code examples was not possible due to the graphical nature of the Blocky interface. Participants were instructed to test out the mapping aids in an example, and the usage of mapping aids was verified in the screenshots and saved data. However, this could be controlled more effectively in a follow-up laboratory study.

Another limitation of the online PLE was that the novices participated on their own computers. For example, the participants had different screen sizes, interactions (touchpad/mouse), and individual settings, including a possibly slow Internet connection, which may also contribute to potential bias. In order to keep possible biases as small as possible, the section of the screen that was viewed was kept constant, regardless of the screen size. Nevertheless, it is possible that the screen size had an influence on the usability or readability of the PDF instructions provided. We recorded screenshots of their progress every 5 seconds to gain insight into the learning behavior of the novices. Although we gained valuable information on their learning, future studies could collect more data points to ensure that no important steps are missing.

Our study extends and uses a preexisting programming environment by Witte and Tichy (2019). Many established alternatives and possibilities to present the mapping aids to the user exist but are out of scope of this work and, therefore, are not systematically evaluated or considered in the present study. Instead, a prototypical implementation is used that employs markers and visualizations existing in ROS. ROS markers are commonly used and representative for visualizing robotic applications in real-world scenarios. This potentially sub-optimal implementation regarding the interaction and visualization of our mapping aids might lead to a reduced effect size in our results.

In our study, we collected a convenience sample. The main goal in recruiting was to find people with little programming experience (novices). Based on the collected data of prior knowledge, the sampling method used succeeded in finding novice programmers. Nevertheless, the sample mainly contained students. Psychology students were overrepresented due to their lower programming experience in comparison to, e.g., engineering students. In this sample, it can be observed that, for example, the figural intelligence was slightly

above average. It would therefore be interesting to see whether the present findings can be replicated with other more heterogeneous samples and what role further learner characteristics (e.g., age, working memory capacity) play in the positive effect of the mapping aids on performance and their use in the PLE. Some of the reported effect sizes were significantly smaller than the initially assumed effect size on which our a priori power analysis was based. The initial sample of the study was also slightly smaller than the sample size proposed by our a priori power analysis. This could also have influenced the effects found. In accordance with good scientific practice, no additional data were collected. In the future, larger samples could be obtained outside the pandemic period, as it was extremely difficult to find subjects during this time. Additionally, support conditions could be chosen so that expected group differences increase.

In the present study, the initial barrier to starting programming and implementing a solution was kept as low as possible. To this end, some hints and helpful information have been provided in the task descriptions to ease the problem-solving aspect of programming. With the help of these hints, novices could divide the tasks into sub-steps and understand the structure of the task and the flight trajectory to be created. This led to a simplification of the programming tasks to make it possible for novices to solve different tasks in less time and to focus on the implementation and program maintenance aspects. The library of available program blocks and the domain-specific language used was tailored to fit the programming tasks. This reduces the time to introduce all necessary concepts and language constructs. Nevertheless, the environment and language can be easily extended and adapted to real-world applications without adding much complexity. The programming tasks are therefore rather artificial and only reminiscent of real-world scenarios, which would be more complex, time-consuming, and require longer training.

## 7 Related Work

Due to the fact that mapping aids in PLEs are an interdisciplinary research field, similar concepts and ideas are often named differently. Consequently, finding related studies and empirical results is not self-evident. In this section, we will provide insight into related work from both software engineering and learning and instruction research. Publications in the field of software engineering often focus on technical aspects. The learning and instruction perspective focuses on empirical evaluation and attributing results to related cognitive processes.

### 7.1 Live Programming Features in Programming Environments for Robotics

Live / on-line PLEs offer different options to implement supportive elements such as mapping aids. Multiple representations or live programming features are commonly used in programming environments for robotics that are targeted at programming novices. Choregraphe (Pot et al. 2009), the programming environment for the NAO humanoid robot uses a data flow-oriented graphical programming model paired with a poseable preview of the robot, as well as a choreography timeline. Subedi et al. (2021) described that using Choregraphe to program NAO was easier to use for novices compared to Python SDK and thus outlined the beneficial effects of live/online PLEs.

The Whyline prototype (Ko and Myers 2004) is another example of an interactive PLE that tries to outline the connection and ease the mapping for novice programmers. It supports an interrogative debugging interface for the Alice programming environment visualizing answers to specific questions. A small user study showed a significant reduction in debugging compared to identical debugging scenarios without Whyline.

Sketch 'n Sketch (Hempel et al. 2019) uses bidirectional evaluation similar to our PLE to create an interactive visual preview with multiple application cases such as HTML, SVG, and markdown editors using functional domain-specific languages as a linked textual representation.

However, quantitative empirical studies of the usability and learning aspects of these environments are rarely done. Berenz and Suzuki (2014) compare their own declarative behavior specification language with the Choregraphe flow-based programming approach in a user study with 17 participants from a professional or educational background. The study focuses on their alternative programming paradigm *Targets-Drives-Means (TDM)* and uses Choregraphe only as a state-of-the-art baseline and does not consider the underlying processes related to multiple representations or mapping aids.

Do et al. (2019) conducted a study with learners with different amount of prior programming experience using the SWELL tool, which also allows bidirectional mapping from program text to output and includes a graphical preview. They used a very brief measurement of the learning outcome and looked at the affective effects of SWELL compared to a standard development environment for programming novices. They found no hints for a beneficial effect in their data. However, they illustrated the potentials and benefits of further research on this topic.

Other case studies focusing on specific aspects, e.g., block-based robot programming in a learning context for children (Sutherland and MacDonald 2018) underline the importance of instant, easy-to-understand, and robust feedback. Alvis Live! (Hundhausen and Brown 2007) provides a live algorithm visualization targeted at novice programmers. In a usability study with 21 novice programmers, the authors found evidence that learners benefit from these live visualizations by quickly identifying and correcting errors. In a later empirical study (Hundhausen et al. 2009) found additional hints for a beneficial effect on novices' programming outcomes when allowing direct manipulation of the preview against a textual programming interface control group. Winterer et al. (2020) use Blockly in an industrial robot programming context showing that even complex programs can be expressed in Blockly, and that the Blockly interface provides similar or better understandability or maintainability than traditional flowchart-based visual languages. Gunaratne et al. (2024) and Sutherland (2022) used Blockly to educate children in programming and described a more structured learning experience, improved understanding of basic concepts of web development, and a better learning experience. Price and Barnes (2015) compared block-based interfaces with textual interfaces in a novice programming environment. Although perceived difficulty did not change, participants using the block-based interface spent less time off task and completed the exercises faster. Although the used interfaces often include some form of preview, they do not consider features linking both views bidirectionally or measure the performance impact of the linking of both views. Campusano et al. (2019) evaluated live programming features to program robot behavior through state machines in



an experiment. Contradicting their assumptions, the live environment did not outperform the non-live state-of-the-art baseline, but participants preferred the live environment.

Table 5 summarizes the results for the most relevant robotic PLEs in related studies evaluating their effectiveness.

## 7.2 Evaluations of Mapping Multiple Representations in Interactive Environments

Many different ways to facilitate mapping in interactive PLEs have been reported and evaluated in terms of their effects on cognitive processes and learning success. Some older studies, for example, used graphical techniques to mark the corresponding elements (inter-representational hyperlinks: Brünken et al. 2005; Seufert et al. 2007). In this case, relevant elements within a representation were marked as hyperlinks and users had the possibility to click on these inter-representational hyperlinks to get hints about corresponding elements in other representations. A beneficial effect of this implementation of the element-to-element mapping aid was presented when comparing the results of the experimental group with a control group (Seufert et al. 2007).

More recently, the approach of linking multiple representations, allowing manipulation of a virtual simulation, was studied in more detail. For example, Rey (2011) compared different mapping aids in a study (text fields, scroll bars, or drag-and-drop) as possibilities that allowed the modification of the parameters of a virtual simulation. These three interactive elements were used to dynamically link the different (multiple) representations with the simulation. Hence, they represent an implementation of relation-to-relation mapping aids, as they do not simply outline related elements but make related processes more transparent (Gentner 1983). Here, positive effects of some interactive elements (scrollbars and drag-and-drop) were found on the transfer or application level, but not on the knowledge level compared to the text field condition. More recently, Liao (2023)

**Table 5** Related evaluations of PLEs and their reported results

	Participants	Evaluation	Results	Features
Subedi et al. (2021)	not reported/ novice prog.	comparison	mixed	Preview
Do et al. (2019)	114 students	evaluation	ineffective	Mapping aid
Hundhausen and Brown (2007)	21 novice prog.	usability study	effective	Preview
Hundhausen et al. (2009)	34 novice prog.	controlled experiment	effective	Preview
Berenz and Suzuki (2014)	17 non-programmers	usability experiment	mostly eff.	Blocks
Price and Barnes (2015)	31 6th/7th graders	controlled experiment	mixed	Blocks, Prev.
Hempel et al. (2018)	21 grad./undergrad.	controlled experiment	mixed	Preview
Sutherland and MacDonald (2018)	not reported/ children	observation of usage	mixed	Blocks
Campusano et al. (2019)	10 engineer. students	controlled experiment	ineffective	Blocks, Prev.
Winterer et al. (2020)	-	example case	effective	Blocks

Most of the PLEs are from the robotic domain and target novices

looked at the effects of different mapping aids. They found effects of mapping aids on different aspects of learning outcome: structural and conceptual knowledge. As already outlined earlier (see Section 7.1) by the example of Sketch 'n Sketch, this approach of dynamic linking between a virtual simulation and a textual representation has been further extended (Mayer et al. 2018; Hempel et al. 2019).

Bidirectional linking of multiple representations – similar to the dynamic linking feature in our PLE – has been implemented in some studies as one possibility to implement relation-to-relation mapping aids. In this case, users were able to manipulate any of the related components to cause a change in the respective other representation (Hempel et al. 2019). For example, Patwardhan and Murthy (2017) discovered the beneficial effects of bidirectional dynamic links as mapping aids, not only for interacting with a PLE but also on higher levels of learning outcomes (transfer, application, or comprehension).

These (heterogeneous) findings illustrate that live programming features per se are not an advantage and that it is necessary to obtain more precise information in larger empirical studies on which type of mapping aids are particularly helpful, which underlying processes are influenced by these mapping aids and how, and which boundary conditions and characteristics of novices play a role in this.

## 8 Conclusion

Domain experts, who operate robotic systems, often lack in-depth programming knowledge. Therefore, the design of the programming (and learning) environment as well as support features to help acquire basic knowledge of the system are crucial to successfully program the CPS. Provenance tracking at run-time provides valuable information that can be used in mapping aids using output-directed programming. These mapping aids support novices, e.g., by automatically linking elements of a live preview to corresponding code elements to help solving programming tasks.

In the present study, we examined the effects of a *highlighting* feature that highlights relevant locations in the code when clicking on an element in the preview, and a *dynamic linking* feature, that allowed direct manipulation of the code through the preview, on program correctness and the learning outcome of novice learners. Participants were tasked to implement different quadcopter missions in a block based PLE with a 3D live preview and simulation.

While highlighting was beneficial for program correctness, dynamic linking showed more positive effects on deeper learning. The combination of both aids compared to single aids also revealed positive effects. The analysis of a typical error showed that especially the combination of both helps increase the chance of a correct solution. Traces of strategies, during solving the mission tasks, can already anticipate success in programming and learning. Hence, our findings are more complex than a global summary allows. One needs to have a more differentiated look at the respective tasks and circumstances to be able to answer whether the different support conditions are helpful or not.

We therefore conclude that output-directed programming and provenance tracking at run-time provides a good basis for implementing mapping aids to relate multiple repre-

sentations. Effective help features and visualizations can be quickly implemented in an environment that provides generic provenance information and enables the manipulation of values using output-directed programming.

## 8.1 Future Work

We plan to implement and evaluate more advanced mapping aids and other live programming features that use and showcase the rich information output-directed programming and provenance tracking in PLEs can provide. The generic foundation provided by output-directed programming and dynamic provenance tracking at the language level enables fast iteration and prototyping of innovative interactive presentation and visualization features. We plan a systematic evaluation of different visualization and interaction variants that improve our prototypical implementation in future studies. Other potential features include improvement of proposed code changes through context information, live evaluation and live debugging tools that can explain program behavior using provenance traces, and test case generation or automatic program repair using assertions that trigger output-directed program changes. For example, the PLE could automatically propose multiple possible fixes for a failed assertion during testing. Improved debugging tools could then provide further insight into how these possible fixes work and whether negative side effects might occur.

More adaptive PLEs could be developed that additionally detect and reinforce positive patterns with prompts and specifically support novices for instance by using AI-features to automatically prune provenance traces based on the current user focus and context. Output-directed programming is a promising and adequate way to provide the necessary information to implement those novel help features.

In future studies, the research question could be extended and the concept of the planning process could be explored in more detail, for example, in connection with the concept of computational thinking (e.g. Hodhod et al. 2014). In this way, the present findings could be integrated into a larger context.

Psychophysiological measurements (e.g., eye-tracking) could provide further insight into the associated cognitive processes. Other formats such as video or audio recordings could be used as possible sources of information. For example, learners could be actively encouraged to verbalize their thoughts and solution processes (think aloud), possibly providing further insight into the strategies used. This could reduce speculation and uncertainty while classifying and interpreting the intent of the participant and thus improve the transparency of behavioral patterns and cognitive processes.

In our study, we found a positive relationship between traces of cognitive learning strategies (organizing, elaborating) and performance based on process data. In future studies, indicators of these traces could be tracked online using learning analytic approaches and used as a live feedback tool to provide helpful inputs (e.g., cognitive prompts) that could trigger the use of learning strategies when needed (e.g. Zumbach et al. 2020). For instance, by recording the time of a novice on a specific task, after a certain amount of time additional information or strategic help could be provided in a pop up window. In this way, individual and adaptive PLEs could be implemented to provide

feedback or appropriate supporting elements. For this, the analysis of further typical errors could be helpful. If adaptation is conceptualized at different levels (micro and macro) and over longer periods of time, further insight could be gained into whether, for example, these factors are also important for finding effects of met-acognitive strategy traces.

In general, our study offers only limited temporal insight. It would also be interesting to observe novices in the process of learning to program for a longer period of time. Investigating whether the effect of the mapping aids and their use changes over time and to what extent the learning effects found are long-term could give insights into how output-directed programming features could help novices become more advanced programmers.

The display concept of the study could also be expanded to non-simulation settings: we developed, both, augmented and virtual reality displays for our PLE. In future studies, we plan to investigate how mapping aids assist novices when the 3D preview is presented as an augmented reality element in the real laboratory, or whether presenting our PLE in virtual reality interferes with the positive effects of our mapping aids.

## Appendices

### A Supplementary Information for the Method

The reported findings are part of a bigger research project and thus not all variables are reported in the paper. Cognitive load as well as the subjective evaluation were not in the focus of this paper and further about the related findings were reported in an other publication (Witte 2024). To measure cognitive load, we used the differentiated cognitive load questionnaire (Klepsch et al. 2017). This questionnaire contained 2 items for intrinsic cognitive load (ICL), reflecting the complexity and element interactivity of the content. We used 3 items for assessing extraneous cognitive load (ECL), which reflects the load imposed by the design of the environment. To test the resources novices invested when programming the three tasks 3 items for germane cognitive load (GCL) were given. The respective items were subjectively rated on a 7-point Likert scale ranging from 1 (not at all) to 7 (completely). Results are reported as z-scores for RQ4. The reliability was sufficient with  $\alpha = .783$  ( $CI_{95\%} = .670 - .861$ ) for ICL, McDonald's omega of  $\omega = .876$  ( $CI_{95\%} = .829 - .923$ ) for ECL, and McDonald's omega of  $\omega = .702$  ( $CI_{95\%} = .595 - .808$ ) for GCL.

## B Programming Exercises

The following pages in this section contain a translated version of the complete instructions and exercises given to the participants preserving the original formatting and layout. Parts that were only included in the instructions for specific experimental conditions are marked accordingly.

### Introduction

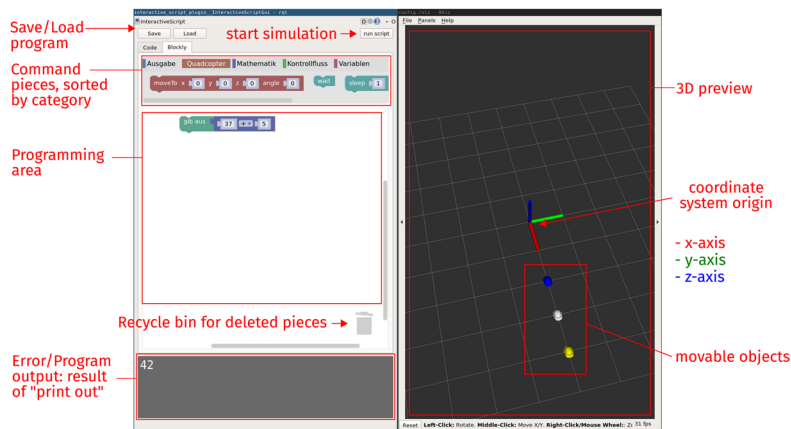
Today you are supposed to program quadcopter missions using a special programming environment for novice programmers. Please read the following pages carefully before you start solving the tasks.

hint

Create a relaxed environment! It is recommended to use a large screen and a mouse and to avoid problems while working on the tasks.

### Programming and learning environment

The main window is divided into several sections: In the programming window pieces from the (expandable) library can be dragged. Below, a preview of the program output or possible errors is displayed. To the right of it a live preview of the program is displayed.



The pieces can be placed anywhere within the programming area, but it is recommended to form a coherent block that is processed from top to bottom. Pieces can be copied (duplicated) by right-clicking on them. Deleted pieces end up in the recycle bin. By clicking on the recycle bin, pieces can be dragged back to the program area.

### 3D preview

The 3D view shows a live preview of the current program. The view can be moved, rotated and zoomed with the mouse. The view can be rotated with the left mouse button, zoomed with the scroll wheel and panned around with the middle mouse button (click with the mouse wheel).

Feel free to try it out!

Waypoints are displayed as small white quadcopters, the path between them as a straight line. If the line is red, the quadcopter may not have enough time to reach the destination (for example, because the next waypoint is given directly without waiting pause), if it is white, the destination is reached.

In addition, the virtual space contains 3 colored objects that can be freely moved and rotated on the floor with the mouse.

### Interactive preview

*Note: This following section was only available in the respective experimental conditions.*






*For conditions with highlighting:*

Clicking on a waypoint will highlight in the programming window the values used to calculate the position of the waypoint. Changing one of the marked fields will move the waypoint, all other values have no influence on the selected waypoint.

*For conditions with dynamic linking:*

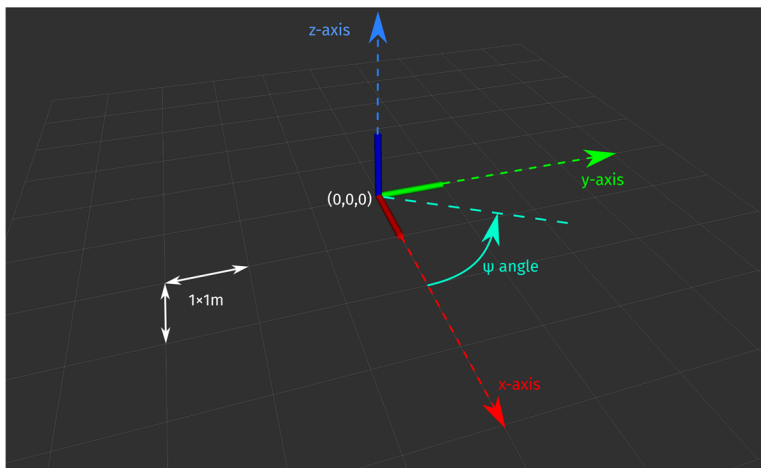
With the arrows a waypoint can be moved directly. It will automatically adjust the program so that the waypoint is located at the desired position and the changes are marked. However, there may be other ways to get the desired result or the change may have other, additional effects.

## Important programming commands

	Writes the value of the right part to the console (lower part of the window)
	Determine new positions on x, y, z (in m) and rotation (in °, around the z-axis)
	Wait until the last set target position has been reached
	Wait for 1 second
	Returns the current position and rotation of an object. Object can be 'hardhat_yellow', 'hardhat_white' or 'hardhat_blue'. The returned pose can be accessed with 'get ... of ...' to get a single value.

## Coordinate System

Positions are always given in meters relative to the origin. This origin is located in the center of the grid. The grid lines form squares of 1x1m and mark the floor of the room. The x-axis is marked in red, the y-axis in green and the z-axis (altitude) in blue. The rotation around the z-axis is measured counterclockwise; 0° correspond to the direction of the x-axis.



## 8 hints for solving the problems

### 1. Programming is an iterative process

It is difficult to find the correct solution directly in the first attempt. Instead, try to create an incomplete solution first and improve it step by step. For example, if you are not sure about the position of the waypoints, it can help to place them somewhere first and then move them to the correct position using the preview.

### 2. Experimentation is encouraged

Simulated quadcopters are cheap and can't break. It is explicitly recommended to try things even if you don't know exactly what will happen. The preview shows the expected program progress at any time. Additionally, by clicking on the 'run script' button, the program can also be simulated.

### 3. Divide and conquer!

If the overall problem seems too complicated, try to identify and solve sub-problems. If these are still too complicated, break the problems down further.

### 4. Display intermediate results

Any intermediate results can be displayed at any time with a 'print out' block. Instead of speculating whether a value is calculated incorrectly, it can simply be output and checked.

### 5. Name intermediate results

Use variables to store intermediate results and to use them multiple times. You can also give them a name: don't be too shy to name the calculated side length of a square 'side length'.

### 6. Take notes

A small drawing or note can be of great help. You don't have to solve everything on the computer.

### 7. Use the preview

The preview is the perfect tool to check programs for errors. It's much faster to figure out where the point (2,1,3) is in space if you just set a waypoint to that position and look it up.

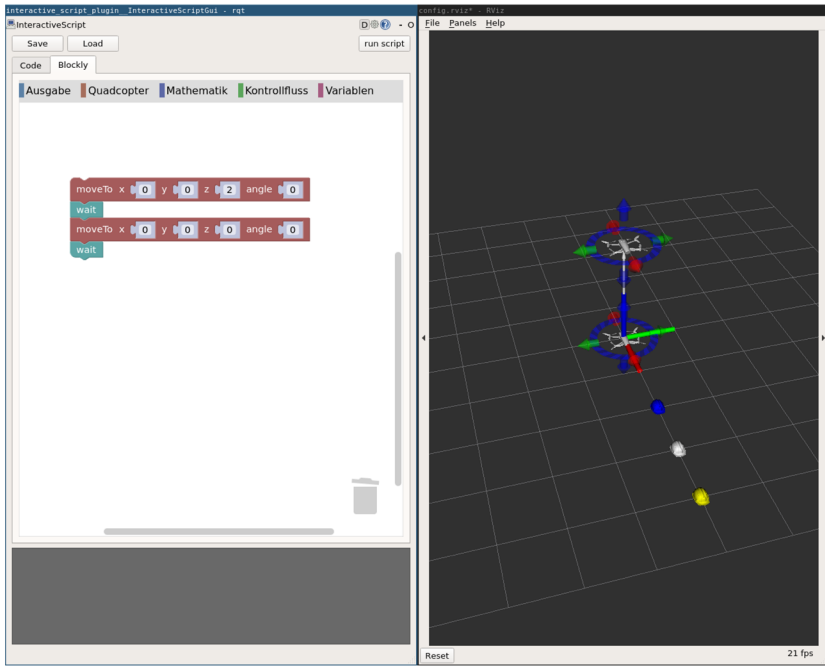
### 8. Pay attention to hint boxes

The tasks contain hint boxes in many places that give useful clues to the solution. It doesn't hurt to read these hints several times if you feel stuck.



## Tutorial example [10min]

The quadcopter is supposed to fly up to a height of 2m above the origin and land again.



### TODO:

- ☐ Replicate the given program.
- ☐ Look at the preview and make sure the waypoint is at 2m height.
- ☐ Run the simulation by clicking on the 'run script' button and compare the simulation with the preview.
- ☐ Try moving the waypoints in the x-direction and experiment with more waypoints or omitting the wait blocks and how that affects the simulation.
- ☐ When you are done click on 'Save' to mark the configuration as a solution and save it (a filename is automatically chosen and there is no feedback! The load function always automatically loads the last save state).

## Mission 1: Simple Square [10min]

The quadcopter is supposed to take off, fly over the four vertices of a square and then land at the coordinate origin (0,0,0). To do this, familiarize yourself with the coordinate system and determine the four corner points of any square in flight space.

hint

It is much easier to align the square to the axes of the coordinate system.

Create a program that makes the quadcopter fly over the four corner points one after the other. You already know the necessary commands from the previous example.

hint

Note that after each new waypoint, it is necessary to wait so that the quadcopter also has the opportunity to reach the target!

Rotate the quadcopter at each corner point so that it is headed towards the center of the square.

hint

*Note: Only available in the respective experimental conditions.*

*For conditions with highlighting:*

You can click on waypoints to mark the corresponding places in the code.

*For conditions with dynamic linking:*

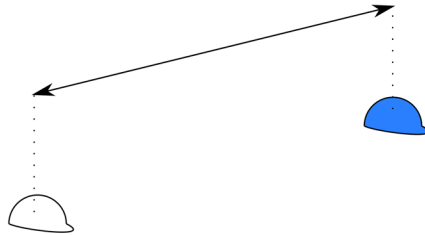
Waypoints in the preview can be moved directly.

### TODO:

- ☐ Determine the corner points of the square.
- ☐ Create a program to fly off the square.
- ☐ Rotate the quadcopter to the center at each corner point.
- ☐ Click on 'Save' to save the program.

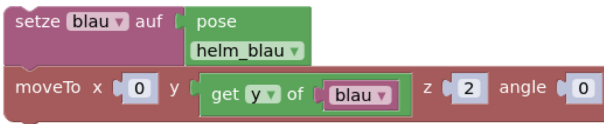
## Mission 2: Commute Between Objects [15min]

Until now, the waypoints were fixed, but now the quadcopter is supposed to fly back and forth between (and above) the positions the white and blue helmet. This allows the quadcopter's flight trajectory to be changed during flight.



The `pose()` function is special because it returns a result (the pose of the object at the time of the call). This result contains several components (x, y, z, angle) which can be accessed by the 'get ... of ...' block from the quadcopter area (e.g. get x of pose('hardhat\_blue') for the x-coordinate of the blue helmet). As an alternative, the result can also be cached and the individual components can be accessed later, i.e.:

hint



To create a variable, select 'Create Variable...' and give it a name. Matching program parts will be created to assign a value to the variable and use the value. Any number of variables can be created as long as they have a unique name without special characters.

hint

Remember that you can output values textually in the gray console area with the 'print out' block. This is useful, for example, to display and check intermediate results.

hint

In order to avoid writing out repeatedly executed commands each time, a loop can be used. The commands within the loop are executed as often as specified in the loop header, e.g.



hint

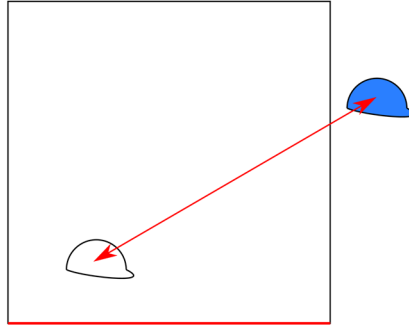
The position of the helmet cannot be used directly, of course, because it is lying on the ground, so the z-coordinate should either be fixed or placed a bit above the helmet.

### TODO:

- ☐ Write a program that determines the pose of the helmets.
- ☐ Calculate the waypoints of the quadcopter from the pose of the helmets.
- ☐ Write a loop so that the waypoints are flown to again and again in turn.
- ☐ Click on 'Save' to save the program.

### Mission 3: Dynamic Square [20min]

Now a little challenge: again a square is supposed to be flown off, but the length of its sides is supposed to be defined by the distance between the white and blue hardhats. For example, if the blue and white helmets are 1.63m apart, the length of an edge of the square should also be 1.63m.



hint

One point of the square can be fixed, the other corner points have to be calculated based on the measured distance.

hint

The distance between two points  $A$  and  $B$  in space is  $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2}$ . Corresponding blocks for the mathematical operations (e.g. square root) can be found in the math section.

hint

It is a good idea to assign the distance to a variable first in order to be able to use it several times.

hint

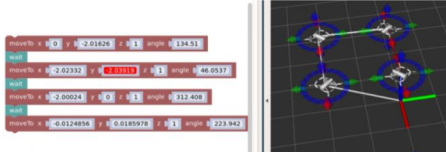

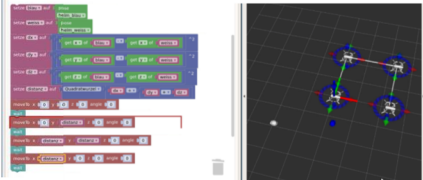
Try to solve the two problems (calculating distance and making the edge length of the square variable) separately: the vertices can be calculated depending on a variable that is assigned a fixed value as a test. Only when you are confident that the correct vertices are calculated for different distances, you should try to calculate the distance between the helmets and use the calculated value.

#### TODO:

- ☐ Write a program that calculates the vertices of a square of variable side length.
- ☐ Write a program that calculates the distance between the hardhats.
- ☐ Use the calculated distance to set the side length of the square.
- ☐ Click on 'Save' to save the program.

## C Solution Scheme for Program Correctness

In this subsection the detailed solution scheme for scoring each of mission is provided including an example of a correct solution.

Examples for a correct solution for each mission	Differentiated scoring scheme providing points (P) for the different substeps of the missions
<p><b>Mission 1</b></p> 	<ul style="list-style-type: none"> <li>- 1P = drone flies (<math>z &gt; 0</math>)</li> <li>- 1P = creating 4 different waypoint</li> <li>- 1P = the waypoints represent a square</li> <li>- 1P = after each „moveTo“ a „wait“/“sleep(suited time)“ follows</li> <li>- 1P = adaption of the angle</li> <li>- 1P = correct adaption of the angle</li> </ul>
<p><b>Mission 2</b></p> 	<ul style="list-style-type: none"> <li>- 1P = „pose“ block related to one of the helmets (blue/white) used</li> <li>- 1P = x defined by „get „x“ of“ „helmet (blue/white)“ block depending on one of the helmets</li> <li>- 1P = y defined by „get „y“ of“ „helmet (blue/white)“ block depending on one of the helmets</li> <li>- 1P= drone flies either defined by a z above the helmet (by certain value or adding a value to the z of a helmet)</li> <li>- 1P = Realting the second helmet to x/y and z (optional)</li> <li>- 1P = adding a loop</li> </ul>
<p><b>Mission 3</b></p> 	<ul style="list-style-type: none"> <li>- 1P = Creating a variable for the side length</li> <li>- 1P = Calculation of the distance of the helmets</li> <li>- 1P = Usage of the „pose“ &amp; „get of“</li> <li>- 3P = Defining the four waypoints depending on the distance of the helmets</li> </ul>

## D Assessment of Prior Knowledge and Learning Outcome

In this subsection all questions related assessing to prior knowledge before the experiment and learning outcome after the missions are presented including the examples for correct answers and scoring.

### D.1 Assessment of Domain-Specific Prior Knowledge – Performance Test

*Please answer the following questions about programming.*

Question	Answer	Points (P)
<i>Knowledge</i>		
What does the abbreviation GUI stand for?	Graphical User Interface	1P
Briefly describe what debugging means.	e.g. Troubleshooting/ Find the cause of the error	1P
What are logical operators? Give an example.	Logical operators are operators that check whether certain statements are true or false have truth value as input and as output. Example: or, and, not	2P
What are the coordinates of the point A ( <i>displayed in a graphic</i> )? Answer example: A(0,2,1)	A(3,4,5)	1P
<i>Comprehension</i>		
Name one major difference between "program execution" and "compilation".	Compile: Translate into computer language, when executed, no translation	1P
Name one major difference between dynamically and statically typed programming languages.	Dynamic: type of a variable can change, not with static.	1P
Name one major difference between variables and values.	A variable is the name of a piece of information, while value is the information itself.	1P
A drone is flying in space and is located at point D(x=1,y=2,z=4). Which coordinate changes when the drone changes its altitude?	Z	1P

### D.2 Subjective Self-Assessment Programming – Self-Report

*Please answer the following questions regarding your previous programming experience.*(Q1) How often have you programmed in the past? (*6-pt Likert scale from daily to never*)

(Q2) Which graphical programming languages have you used before?

(Q3) Which other programming languages have you used before?

(Q4) How do you rate your current knowledge of each of the following programming concepts?

- (*6-pt Likert scale from no knowledge to very knowledgeable*)Handling of variables
- Handling of data types
- Flow control with loops
- Flow control with conditions
- Function definition

### D.3 Assessment of Learning Outcome – Performance Test

Please answer the following questions.

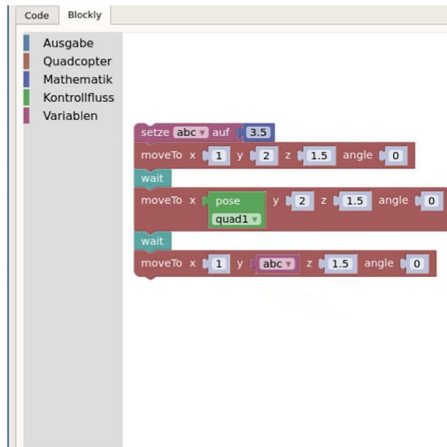
Question	Answer	Points (P)
<b>Knowledge</b>		1P
What does the "wait" command do?		
a) The program waits for further input and if there is no further input it terminates after 5 seconds.	c) Stops the program until the drone has reached the waypoint defined afterwards.	
b) Stops the program until the drone has reached the previously defined waypoint.		
c) Stops the program until the drone has reached the waypoint defined afterwards.		
d) The drone stops immediately and waits there for further commands.		
Which statements about variables are correct?		1P
a) Only numbers and words can be assigned as values.		b) A variable can be assigned multiple times.
b) A variable can be assigned multiple times.		c) A variable can always be used instead of the value assigned to it.
c) A variable can always be used instead of the value assigned to it.		
d) The value of a variable is unchangeable.		
Which statements about flight maneuvers of the drone are correct?		1P
a) The drone can fly in any direction regardless of its orientation.	a) The drone can fly in any direction regardless of its orientation.	
b) The angle of the "moveTo" block indicates the rotation around the z-axis.	b) The angle of the "moveTo" block indicates the rotation around the z-axis.	
c) The angle of the "moveTo" block indicates the rotation around the y-axis.		
d) The angle of the "moveTo" block indicates the rotation around the x-axis.		
<b>Comprehension</b>		
Name a basic difference between the "pose" block and the "wait" block.	Pose: expression; returns a value, does not block the program, expects a parameter (which object to assign); Wait is a statement	2P
Describe the effect of the 3 programs a,b,c on the movement of the drone in your own words.		3P
a) moveTo – wait – sleep – moveTo	a) D flies to point 1, program is stopped until the point is reached. D waits at point 1 t (in sleep defined time) and then flies to P2.	
b) moveTo – wait – moveTo	b) D flies to point 1, program is stopped until the point is reached. D waits at point 1 t (in sleep defined time) and then flies to P2.D flies to P1, when reached program continues to run, D flies to P2	

c) moveTo – sleep – wait – moveTo

c) D flies to P1 if sleep time already elapsed, then no effect; otherwise program waits at P1 for remaining time, wait ensures that point is approached in any case, P2 is approached

### Application

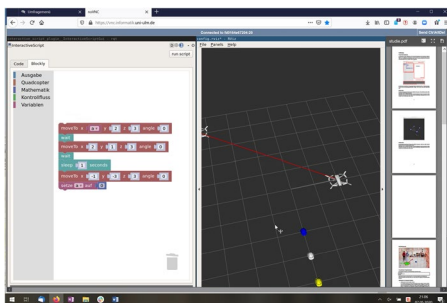
Name the problem in this program:



Pose returns a structure with multiple fields, only one component of this pose can be inserted into the moveTo block like this

1P

Name the problem in this program:



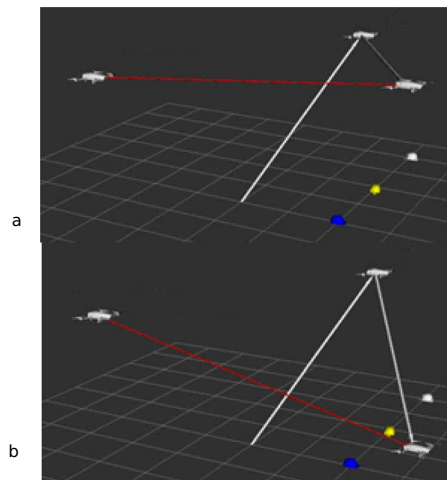
Variable used first and only then value assignment/definition

1P

What is the difference between the code for preview a and preview b?

Second waypoint: z-coordinate has been altered

1P

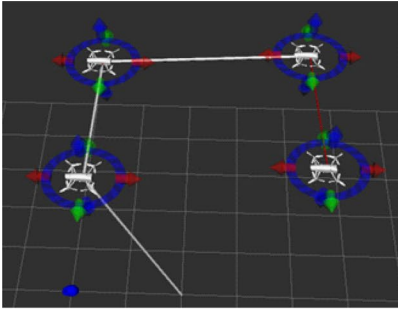




Which program matches this preview?

a

1P



a

```
moveTo x 1.5 y 0 z 3 angle 0
wait
moveTo x 1.44755 y -2.72487 z 3 angle 0
wait
moveTo x -2.10285 y -3 z 3 angle 0
wait
moveTo x -2.244 y -0.320811 z 3 angle 0
```

b

```
moveTo x 1.5 y 0 z 3 angle 0
wait
moveTo x 1.44755 y -2.72487 z 3 angle 0
wait
moveTo x -2.10285 y -3 z 3 angle 0
wait
moveTo x 5 y -0.320811 z 3 angle 0
```

c

```
moveTo x 1.5 y 0 z 3 angle 0
wait
moveTo x 1.44755 y -2.72487 z 3 angle 0
wait
moveTo x -2.10285 y -3 z 0 angle 0
wait
moveTo x 5 y -0.320811 z 3 angle 0
```

d

```
moveTo x 1.5 y 0 z 3 angle 0
wait
moveTo x 1.44755 y -2.72487 z 3 angle 0
wait
moveTo x 4 y -0.320811 z 3 angle 0
```

## E Typical Errors while Programming the Three Missions

In the following, we provide more insights into typical errors while programming the first mission. Based on our evaluation scheme, we analyzed individual errors that occurred. In general, the following errors were considered in the evaluation scheme:

- quadcopter is flying ( $z > 0$ )
- creating 4 waypoints
- waypoints result in a square
- at least 1x change of an angle
- correct angle setting

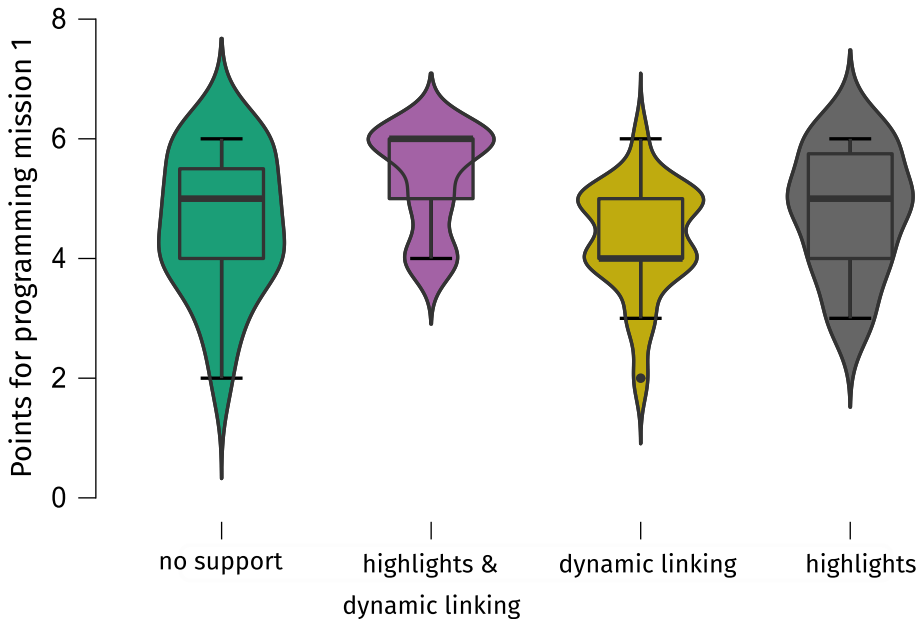


Fig. 11 Violin plot displaying the point distribution for the first mission

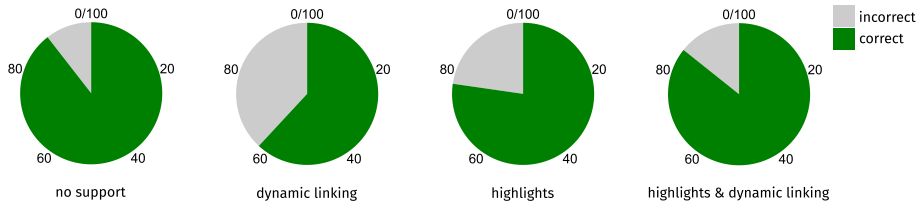
- each *moveTo* is followed by a *wait* or *sleep* block Overall, novices could achieve 6 point maximum. Only 8 of 82 scored 3 points or below in the first mission (see Fig. 11).

Based on the violin plot displaying the descriptive pattern of the points in the first mission for each experimental group, differences can be observed between the four experimental groups (see Fig. 11).

Descriptively, most novices scoring 3 points or lower were found in the control group, followed by the group with dynamic linking as mapping aid. By analyzing the process data, including the automatically recorded screenshots during programming the mission, typical errors have been determined by two independent raters. Around 80 % of the novices scored with 4 points or higher. Mostly, errors resulting in lower scores occurred based on three of the six evaluation criteria: the drone does not fly ( $z = 0$ ), the angles of the waypoints were not manipulated at all, and the angles were altered but the angle setting was not correct. The last aspect is described as an exemplary, typical error in the results section (see Section 5.7). The other two typical errors are described in more detail in the following sections including a descriptive overview along with inferential testing using logistic regression approaches.

### E.1 Quadcopters didn't Fly

We analyzed the typical error not of altering the z-coordinate resulting in a quadcopter that does not fly. Descriptively, novices in the group with dynamic linking and/or highlights were less likely to alter the z-coordinate (see Fig. 12). The lowest descriptive probability of not altering the z-coordinate was found for the experimental group with dynamic linking.



**Fig. 12** Pie charts displaying the proportion of drones that didn't fly (incorrect) and the flying drones (correct) for each experimental group

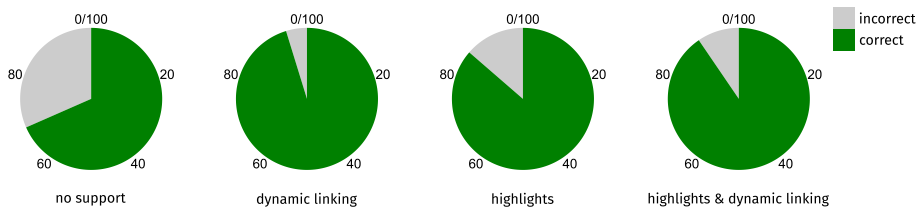
**Table 6** Coefficients

	Estimate	Std. Error	Odds Ratio	z	Wald Stat.	Wald Test	
						df	p
(Intercept)	2.343	1.650	10.417	1.420	2.016	1	0.156
Dyn. linking * highlights	-0.376	0.982	0.687	-0.383	0.147	1	0.702
Dynamic linking	-1.790	0.891	0.167	-2.010	4.039	1	0.044
Highlights	-1.038	0.918	0.354	-1.130	1.278	1	0.258
Figural intelligence	1.303	1.773	3.682	0.735	0.540	1	0.462
Need for cognition	-0.007	0.018	0.993	-0.383	0.147	1	0.702
Prior knowledge	-0.010	0.016	0.990	-0.646	0.418	1	0.518

To determine, whether this trend represents a significant effect, we conducted a logistic regression considering respective covariates. Our analysis revealed, that the dynamic linking group had a significant lower probability to adjust the z-coordinate compared to the control group (see Table 6). This finding might support the idea described in the theory and previous literature, that the beneficial effect of mapping aids is also dependent of the considered (sub)task.

## E.2 No Angle Adjustment

Next to the described angle setting error in the results section (see Section 5.7), we additionally analyzed whether the provided mapping aids had an impact on the odds of changing the angle at least once. Descriptively, in the control group without mapping aids, novices were more likely to not adjust the angle at all (see Fig. 13).



**Fig. 13** Pie charts displaying the proportion of no adaption of the angles (incorrect) and at least once adapted angles of the drones (correct) for each experimental group

**Table 7** Coefficients of the logistic regression predicting the chance of at least one angle adaption in mission 1 considering the effect of experimental groups (1-3) against the control group (4)

	Estimate	Std. Error	Odds Ratio	z	Wald Stat.	df	Wald Test
							p
(Intercept)	-1.581	1.768	0.206	-0.894	0.800	1	0.371
Dyn. linking * highlights	2.218	1.152	9.189	1.925	3.707	1	0.054
Dynamic linking	2.351	1.168	10.495	2.013	4.050	1	0.044
Highlights	1.281	0.840	3.601	1.525	2.327	1	0.127
Figural intelligence	0.842	2.120	2.320	0.397	0.158	1	0.691
Need for cognition	0.026	0.021	1.026	1.249	1.560	1	0.212
Prior knowledge	0.005	0.020	1.005	0.260	0.067	1	0.795

We tested this typical error for significance by using a logistic regression considering different covariates: Significant effects of the mapping aids for both mapping aids and the dynamic linking were found (see Table 7). Novices in these groups were more likely to adapt the angle at least once compared to the control group, when the respective covariates were considered.

**Author Contributions** TW and AV developed the initial design of the study (Conceptualization) which was feed backed by TS and MT (Supervision). AV and TW developed the missions and the study specific questions (learning outcome, prior knowledge etc.; Methodology). TW implemented the PLE and managed the technical infrastructure (Software; Visualization). AV led the data collection for the study (Investigation). AV and TW analyzed and interpreted the data (Formal analysis). AV and TW drafted the work (Writing – Original Draft), which was revised critically by MT and TS (Writing - Review & Editing). All authors provided approval of the final submitted version of the manuscript and agree to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

**Funding** Open Access funding enabled and organized by Projekt DEAL. This work was partially supported by the German Research Foundation (DFG): 435878599, 453895475, and the German Federal Ministry of Education and Research (BMBF): 16DHB2205.

**Data Availability** The data supporting the findings of this study are publicly available at doi.org/10.5281/zenodo.10908067. The source code and scripts to run the programming and learning environment used in the study are publicly available on our github page at [https://github.com/sp-uulm/interactive\\_script/release\\_s/tag/StudyPLE](https://github.com/sp-uulm/interactive_script/release_s/tag/StudyPLE).

## Declarations

**Ethical Approval** The study was carried out in accordance with the Declaration of Helsinki. Ethical review and approval were not required for the study on human participants in accordance with the local legislation and institutional requirements.

**Informed consent** The participants provided their written informed consent to participate in this study. The data were used pseudonymously, and participants were aware that they had the chance to withdraw their data at any point in the study.

**Conflicts of interest** The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

**Clinical Trial Number** Not applicable.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as

you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Acar UA, Ahmed A, Cheney J, Perera R (2013) A core calculus for provenance. *J Comput Secur* 21(6):919–969
- Ainsworth S (1999) The functions of multiple representations. *Comput Educ* 33(2–3):131–152. [https://doi.org/10.1016/S0360-1315\(99\)00029-9](https://doi.org/10.1016/S0360-1315(99)00029-9)
- Ainsworth S (2014) The multiple representation principle in multimedia learning. In: Mayer RE (ed) *The Cambridge handbook of multimedia learning*, Cambridge Handbooks in Psychology, Cambridge University Press, New York, pp 464–486. <https://doi.org/10.1017/CBO9781139547369.024>
- Becker RA, Cleveland WS (1987) Brushing scatterplots. *Technometrics* 29(2):127–142
- Beißert H, Köhler M, Rempel M, Beierlein C (2014) Eine deutschsprachige kurzsкала zur messung des konstrukts need for cognition: Die need for cognition kurzsкала (NFC-K), GESIS-Working Papers, vol 2014/32. GESIS - Leibniz-Institut für Sozialwissenschaften, Mannheim, <https://nbn-resolving.org/urn:nbn:de:0168-ssao-403157>
- Berenz V, Suzuki K (2014) Targets-drives-means: A declarative approach to dynamic behavior specification with higher usability. *Robotics Auton Syst* 62(4):545–555. <https://doi.org/10.1016/j.robot.2013.12.010>
- Bloom BS, Engelhart MD, Furst E, Hill WH, Krathwohl DR (1956) *Taxonomy of educational objectives. Vol. 1: Cognitive Domain*. David McKay, New York
- Breckel A, Tichy M (2016) Live programming with code portals. In: *Workshop on live programming systems (LIVE'16)*, Rome, Italy
- Brünken R, Seufert T, Zander S (2005) Förderung der Kohärenzbildung beim Lernen mit multiplen Repräsentationen. *Z Pädagog Psychol* 19(1/2):61–75. <https://doi.org/10.1024/1010-0652.19.12.61>
- Campusano M, Fabry J (2017) Live robot programming: The language, its implementation, and robot api independence. *Sci Comput Progr* 133:1–19. <https://doi.org/10.1016/j.scico.2016.06.002>
- Campusano M, Fabry J, Bergel A (2019) Live programming in practice: A controlled experiment on state machines for robotic behaviors. *Inf Softw Technol* 108:99–114. <https://doi.org/10.1016/j.infsof.2018.12.008>
- Jong T, Joolingen WR (1998) Scientific discovery learning with computer simulations of conceptual domains. *Rev Educ Res* 68(2):179–201. <https://doi.org/10.3102/00346543068002179>
- Do Q, Campbell K, Hine E, Pham D, Taylor A, Howley I, Barowy DW (2019) Evaluating prodirect manipulation in hour of code. In: *Proceedings of the 2019 ACM SIGPLAN symposium on SPLASH-E*, pp 25–35
- Erhel S, Jamet E (2019) Improving instructions in educational computer games: Exploring the relations between goal specificity, flow experience and learning outcomes. *Comput Human Behav* 91:106–114. <https://doi.org/10.1016/j.chb.2018.09.020>
- Faul F, Erdfelder E, Buchner A, Lang AG (2009) Statistical power analyses using G\*Power 3.1: Tests for correlation and regression analyses. *Behav Res Methods* 1(4):1149–1160. <https://doi.org/10.3758/BRM.41.4.1149>
- Field A (2013) *Discovering statistics using IBM SPSS statistics: And sex and drugs and rock 'n' roll*, 4th edn. MobileStudy, SAGE, Los Angeles and London and New Delhi and Singapore and Washington DC
- Finch H (2005) Comparison of the performance of nonparametric and parametric manova test statistics when assumptions are violated. *Methodol* 1(1):27–38. <https://doi.org/10.1027/1614-1881.1.1.27>
- Fincham E, Gašević D, Jovanović J, Pardo A (2018) From study tactics to learning strategies: An analytical method for extracting interpretable representations. *IEEE Trans Learn Technol* 12(1):59–72
- Fisher J (1991) Defining the novice user. *Behav Inf Technol* 10(5):437–441
- Fries L, Son JY, Givvin KB, Stigler JW (2021) Practicing connections: A framework to guide instructional design for developing understanding in complex domains. *Educ Psychol Rev* 33(2):739–762. <https://doi.org/10.1007/s10648-020-09561-x>
- Gentner D (1983) Structure-mapping: A theoretical framework for analogy. *Cognit Sci* 7(2):155–170. [https://doi.org/10.1207/s15516709cog0702\\_3](https://doi.org/10.1207/s15516709cog0702_3)
- Gunaratne M, Weerasekara S, Weerakkody D, Sashmitha N, De Zoysa R, Kodagoda N (2024) Web block craft: web development for children using google blockly. *Int J Electr Comput Eng* 14

- Hadwin AF (2021) Commentary and future directions: What can multi-modal data reveal about temporal and adaptive processes in self-regulated learning? *Learning and Instruction* 72:101287. <https://doi.org/10.1016/j.learninstruc.2019.101287>
- Hadwin AF, Nesbit JC, Jamieson-Noel D, Code J, Winne PH (2007) Examining trace data to explore self-regulated learning. *Metacognit Learn* 2(2–3):107–124. <https://doi.org/10.1007/s11409-007-9016-7>
- Hempel B, Chugh R (2020) Tiny structure editors for low, low prices! (Generating GUIs from toString functions). In: 2020 IEEE symposium on visual languages and human-centric computing (VL/HCC), IEEE, pp 1–5. <https://doi.org/10.1109/VL/HCC50065.2020.9127256>
- Hempel B, Lubin J, Lu G, Chugh R (2018) DEUCE: A lightweight user interface for structured editing. In: Proceedings of the 40th international conference on software engineering, association for computing machinery, New York, NY, USA, ICSE '18, p 654–664. <https://doi.org/10.1145/3180155.3180165>
- Hempel B, Lubin J, Chugh R (2019) Sketch-n-sketch: Output-directed programming for svg. In: Proceedings of the 32nd Annual ACM symposium on user interface software and technology, Association for Computing Machinery, New York, NY, USA, UIST '19, p 281–292. <https://doi.org/10.1145/3332165.3347925>
- Hodhod R, Fleenor H, Nabi S (2014) Adaptive augmented reality serious game to foster problem solving skills. In: Workshop Proceedings of The 10th international conference on intelligent environments, IOS Press, pp 273–284. <https://doi.org/10.3233/978-1-61499-411-4-273>
- Hundhausen CD, Brown JL (2007) What you see is what you code: A “live” algorithm development and visualization environment for novice learners. *J Vis Lang Comput* 18(1):22–47. <https://doi.org/10.1016/j.jvlc.2006.03.002>
- Hundhausen CD, Farley SF, Brown JL (2009) Can direct manipulation lower the barriers to computer programming and promote transfer of training? an experimental study. *ACM Trans Comput-Human Interact (TOCHI)* 16(3):1–40
- Jeske D, Backhaus J, Stamov Roßnagel C (2014) Self-regulation during e-learning: using behavioural evidence from navigation log files. *J Comput Assist Learn* 30(3):272–284. <https://doi.org/10.1111/jcal.12045>
- Kang H, Guo PJ (2017) Omnicode: A novice-oriented live programming environment with always-on runtime value visualizations. In: Proceedings of the 30th Annual ACM symposium on user interface software and technology, association for computing machinery, New York, NY, USA, UIST '17, p 737–745. <https://doi.org/10.1145/3126594.3126632>
- Klepsch M, Schmitz F, Seufert T (2017) Development and validation of two instruments measuring intrinsic, extraneous, and germane cognitive load. *Front Psychol* 1997. <https://doi.org/10.3389/fpsyg.2017.01997>
- Ko AJ, Myers BA (2004) Designing the whyline: a debugging interface for asking questions about program behavior. In: Dykstra-Erickson E, Tscheligi M (eds) Proceedings of the 2004 conference on human factors in computing systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004, ACM, pp 151–158. <https://doi.org/10.1145/985692.985712>
- Kölling M, Brown N, Altmirri A (2017) Frame-based editing. *J Vis Lang Sentient Syst* 3:40–67
- Koytek P, Perin C, Vermeulen J, André E, Carpendale S (2017) Mybrush: Brushing and linking with personal agency. *IEEE Trans Vis Comput Graph* 24(1):605–615
- Kozma RB, Russell J (1997) Multimedia and understanding: Expert and novice responses to different representations of chemical phenomena. *J Res Sci Teach Off J Nat Assoc Res Sci Teach* 34(9):949–968
- Lahtinen E (2007) A categorization of novice programmers: A cluster analysis study. *PPIG* 16:32–41
- Liao H (2023) Mapping principles and worked examples for structural learning: effects of content complexity. *Front Psychol* 14:1241873
- Lister R, Adams ES, Fitzgerald S, Fone W, Hamer J, Lindholm M, McCartney R, Moström JE, Sanders K, Seppälä O et al (2004) A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin* 36(4):119–150
- Lowe RK (1996) Background knowledge and the construction of a situational representation from a diagram. *Eur J Psychol Educ* 11(4):377–397. <https://doi.org/10.1007/BF03173279>
- Malmberg J, Järvelä S, Järvenoja H (2017) Capturing temporal and sequential patterns of self-, co-, and socially shared regulation in the context of collaborative learning. *Contemp Educ Psychol* 49:160–174. <https://doi.org/10.1016/j.cedpsych.2017.01.009>
- Matcha W, Gašević D, Uzir NA, Jovanović J, Pardo A (2019) Analytics of learning strategies: Associations with academic performance and feedback. In: Proceedings of the 9th international conference on learning analytics & knowledge, pp 461–470
- Mayer M, Kuncak V, Chugh R (2018) Bidirectional evaluation with direct manipulation. *Proc ACM Progr Lang* 2(OOPSLA) 1–28. <https://doi.org/10.1145/3276497>
- Mayer RE (1981) The psychology of how novices learn computer programming. *ACM Comput Surv (CSUR)* 13(1):121–141
- Mayer RE (1988) Learning strategies: An overview. *Learn Stud Strateg* 11–22

- Mayer RE, Mathias A, Wetzell K (2002) Fostering understanding of multimedia messages through pre-training: Evidence for a two-stage theory of mental model construction. *J Exper Psychol Appl* 8(3):147. <https://doi.org/10.1037/1076-898X.8.3.147>
- Ozcelik E, Arslan-Ari I, Cagiltay K (2010) Why does signaling enhance multimedia learning? evidence from eye movements. *Comput Human Behav* 26(1):110–117. <https://doi.org/10.1016/j.chb.2009.09.001>
- Pasternak E, Fenichel R, Marshall AN (2017) Tips for creating a block language with blockly. In: 2017 IEEE blocks and beyond workshop (B & B), IEEE, pp 21–24. <https://doi.org/10.1109/BLOCKS.2017.8120404>
- Patwardhan M, Murthy S (2017) Designing reciprocative dynamic linking to improve learners' representational competence in interactive learning environments. *Res Pract Technol Enhanc Learn* 12(1):10. <https://doi.org/10.1186/s41039-017-0046-8>
- Pavlova MV (2024) Building abstraction: The role of representation and structural alignment in learning. In: Proceedings of the annual meeting of the cognitive science society, 46 (0)
- Pot E, Monceaux J, Gelin R, Maisonnier B (2009) Choregraphe: A graphical tool for humanoid robot programming. In: RO-MAN 2009 - The 18th IEEE international symposium on robot and human interactive communication, pp 46–51. <https://doi.org/10.1109/ROMAN.2009.5326209>
- Prather J, Pettit R, McMurtry K, Peters A, Homer J, Cohen M (2018) Metacognitive difficulties faced by novice programmers in automated assessment tools. In: Malmi L, Korhonen A, McCartney R, Petersen A (eds) Proceedings of the 2018 ACM conference on international computing education research, ACM, New York, NY, USA, pp 41–50. <https://doi.org/10.1145/3230977.3230981>
- Price TW, Barnes T (2015) Comparing textual and block interfaces in a novice programming environment. In: Proceedings of the 11th annual international conference on international computing education research, Association for Computing Machinery, New York, NY, USA, ICER '15, p 91–99, <https://doi.org/10.1145/2787622.2787712>
- Qian Y, Lehman J (2017) Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans Comput Educ (TOCE)* 18(1):1–24
- Rey GD (2011) Interactive elements for dynamically linked multiple representations in computer simulations. *Appl Cognit Psychol* 25(1):12–19. <https://doi.org/10.1002/acp.1633>
- Roldán-Gómez JJ, González-Gironde E, Barrientos A (2021) A survey on robotic technologies for forest firefighting: Applying drone swarms to improve firefighters' efficiency and safety. *Appl Sci* 11(1):363. <https://doi.org/10.3390/app11010363>
- Roll I, Winne PH (2015) Understanding, evaluating, and supporting self-regulated learning using learning analytics. *J Learn Anal* 2(1):7–12. <https://doi.org/10.18608/jla.2015.21.2>
- Rossano GF, Martinez C, Hedelind M, Murphy S, Fuhlbrigge TA (2013) Easy robot programming concepts: An industrial perspective. In: 2013 IEEE international conference on automation science and engineering (CASE), IEEE, pp 1119–1126. <https://doi.org/10.1109/CoASE.2013.6654035>
- Ruttun RD, Macredie RD (2012) The effects of individual differences and visual instructional aids on disorientation, learning performance and attitudes in a hypermedia learning system. *Comput Human Behav* 28(6):2182–2198
- Schipolowski S, Wilhelm O, Schroeders U (2017) Berliner Test zur Erfassung fluider und kristalliner Intelligenz ab der 11. Jahrgangsstufe (BEFKI 11+) [Berlin test of fluid and crystallized intelligence for grades 11 and above]
- Schnotz W, Bannert M (2003) Construction and interference in learning from multiple representation. *Learn Instruct* 13(2):141–156. [https://doi.org/10.1016/S0959-4752\(02\)00017-8](https://doi.org/10.1016/S0959-4752(02)00017-8)
- Seal KC, Przasnyski ZH, Leon LA (2010) How levels of interactivity in tutorials affect students' learning of modeling transportation problems in a spreadsheet. *Decis Sci J Innov Educ* 8(1):75–94. <https://doi.org/10.1111/j.1540-4609.2009.00244.x>
- Seufert T (2019) Training for coherence formation when learning from text and picture and the interplay with learners' prior knowledge. *Front Psychol* 10:193. <https://doi.org/10.3389/fpsyg.2019.00193>
- Seufert T, Brünken R (2006) Cognitive load and the format of instructional aids for coherence formation. *Appl Cognit Psychol* 20(3):321–331. <https://doi.org/10.1002/acp.1248>
- Seufert T, Jänen I, Brünken R (2007) The impact of intrinsic cognitive load on the effectiveness of graphical help for coherence formation. *Comput Human Behav* 23(3):1055–1071. <https://doi.org/10.1016/j.chb.2006.10.002>
- Sobral SR (2021) Bloom's taxonomy to improve teaching-learning in introduction to programming. *Int J Inf Educ Technol* 11(3):148–153. <https://doi.org/10.18178/ijiet.2021.11.3.1504>
- Subedi A, Pandey D, Mishra D (2021) Programming nao as an educational agent: a comparison between choregraphe and python sdk. In: The Proceedings of the international conference on smart city applications, Springer, pp 367–377
- Sutherland CJ (2022) Blockly in a box: How children explore block-based robot programming. In: 2022 19th international conference on ubiquitous robots (UR), IEEE, pp 263–267



- Sutherland CJ, MacDonald BA (2018) Naoblocks: A case study of developing a children's robot programming environment. In: 2018 15th international conference on ubiquitous robots (UR), pp 431–436, <https://doi.org/10.1109/URAI.2018.8441843>
- Sweller J (2020) Cognitive load theory and educational technology. *Educ Technol Res Develop* 68(1):1–16. <https://doi.org/10.1007/s11423-019-09701-3>
- Tanimoto SL (2013) A perspective on the evolution of live programming. In: 2013 1st international workshop on live programming (LIVE), pp 31–34, <https://doi.org/10.1109/LIVE.2013.6617346>
- Vainio V, Sajaniemi J (2007) Factors in novice programmers' poor tracing skills. *ACM SIGCSE Bull* 39(3):236–240
- van der Meij J, de Jong T (2006) Supporting students' learning with multiple representations in a dynamic simulation-based learning environment. *Learn Instruct* 16(3):199–212. <https://doi.org/10.1016/j.learninstruc.2006.03.007>
- van Someren MW, Reimann P, Boshuizen H, de Jong T et al (1998) Learning with multiple representations. *Adv Learn Instruct Ser*, ERIC
- Venables A, Tan G, Lister R (2009) A closer look at tracing, explaining and code writing skills in the novice programmer. In: Proceedings of the 5th international workshop on computing education research workshop, pp 117–128
- Vogt A (2021) Fostering deep learning in immersive virtual reality: Interplay of learner's characteristics with internal and external support. PhD thesis, Ulm University. <https://doi.org/10.18725/OPARU-44200>
- Weinstein CE, Underwood VL (1985) Learning strategies: The how of learning. *Think Learn* 1:241–258
- Weintrop D, Afzal A, Salac J, Francis P, Li B, Shepherd DC, Franklin D (2018) Evaluating coblox: A comparative study of robotics programming environments for adult novices. In: Proceedings of the 2018 CHI conference on human factors in computing systems, pp 1–12
- Wild KP, Schiefele U (1994) Lernstrategien im studium: Ergebnisse zur faktorenstruktur und reliabilität eines neuen fragebogens. *Zeitschrift für Differentielle und Diagnostische Psychologie*. <https://psycnet.apa.org/record/1996-85746-001>
- Winne PH, Jamieson-Noel D (2002) Exploring students' calibration of self reports about study tactics and achievement. *Contemp Educ Psychol* 27(4):551–572
- Winterer M, Salomon C, Köberle J, Ramler R, Schittengruber M (2020) An expert review on the applicability of blockly for industrial robot programming. In: 2020 25th IEEE international conference on emerging technologies and factory automation (ETFA), vol 1, pp 1231–1234. <https://doi.org/10.1109/ETFA46521.2020.9212036>
- Witte T (2024) What you change is what you get: using provenance tracking at run-time in component-based robotic applications. PhD thesis, Universität Ulm
- Witte T, Tichy M (2019) A hybrid editor for fast robot mission prototyping. In: 2019 34th IEEE/ACM international conference on automated software engineering workshop (ASEW), pp 41–44. <https://doi.org/10.1109/ASEW.2019.00026>
- Zumbach J, Rammerstorfer L, Deibl I (2020) Cognitive and metacognitive support in learning with a serious game about demographic change. *Comput Human Behav* 103:120–129. <https://doi.org/10.1016/j.chb.2019.09.026>

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



## Authors and Affiliations

Thomas Witte<sup>1</sup> · Andrea Vogt<sup>2</sup> · Tina Seufert<sup>3</sup> · Matthias Tichy<sup>1</sup> 

✉ Matthias Tichy  
matthias.tichy@uni-ulm.de

Thomas Witte  
thomas.witte@uni-ulm.de

Andrea Vogt  
andrea.vogt@dlr.de

Tina Seufert  
tina.seufert@uni-ulm.de

<sup>1</sup> Institute of Software Engineering and Programming Languages, Ulm University, Ulm, Germany

<sup>2</sup> Department Learning and Instruction, Institute of Psychology and Education, Ulm University, Ulm, Germany

<sup>3</sup> Department AI Engineering, Institute for AI Safety and Security, German Aerospace Center (DLR), Ulm, Germany