Wizard or Example? Supporting Robot Program Reuse Discovery Across Expertise Levels

Yoganata Kristanto[1], Mina Alipour[1], Miguel Campusano[1] and Aljaz Kramberger[2]

[1]SDU Software Engineering, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Campusvej 55, Odense, 5230, Denmark

[2]SDU Robotics, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Campusvej 55, Odense, 5230, Denmark

Abstract

Robot program reuse has been a challenge because it is tightly bound to the configuration it is designed for. This increases implementation costs and complexity. We propose a search tool that allows users of all levels to discover relevant robot programs through wizard-guided and example-based search methods. In this work, we investigate how interface design and domain expertise interact to shape search behavior in a machine tending robotic application. This investigation is conducted through a user study involving 10 participants across two levels of expertise (novice, expert), each completing two search tasks. Our findings show that participants have a higher success rate using the wizard-guided method than the example-based method; however, experts mention preference for example-based method for its finer grain control, while novices prefer the other for their familiarity with parameter configuration.

Keywords

Collaborative robot, Search Interface, Program Reuse

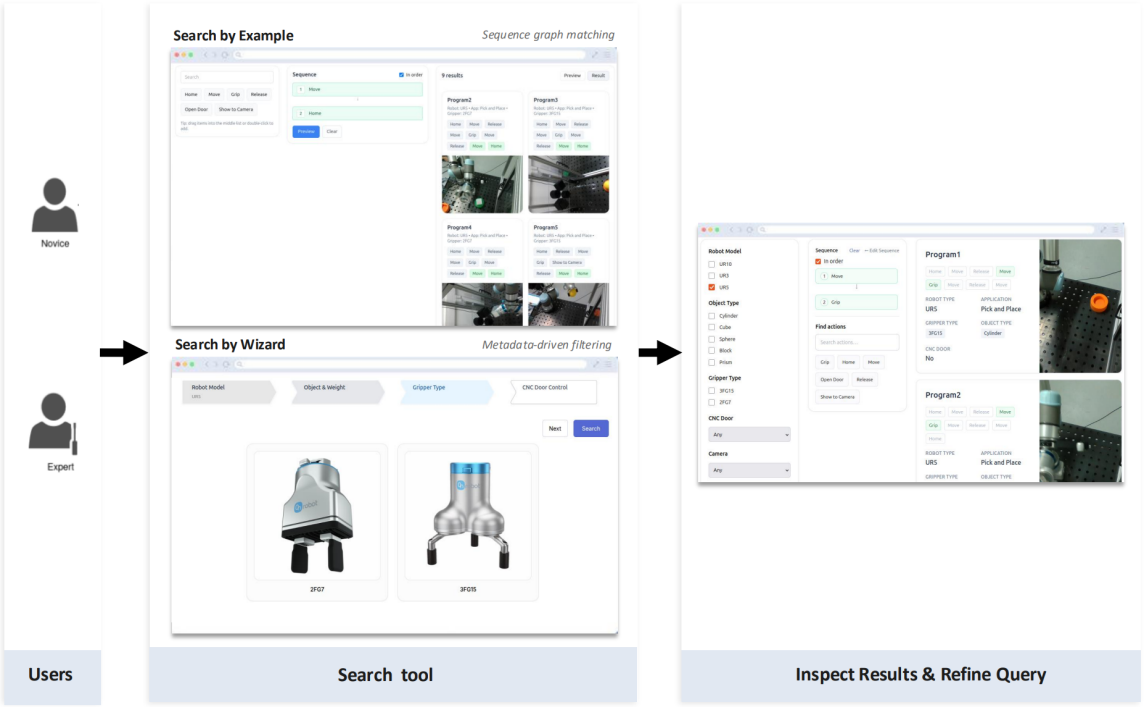


Figure 1: Search Tool Overview

1. Introduction

Robotics software development often relies on reusing program modules across different tasks and hardware setups. However, program reuse remains difficult because existing code is often tightly bound to the specific configurations for which it was originally designed. Such as: Hardware (robot model, end-effector, sensors), Physical Constraints (workspace geometry, reachability), and Task-Specific Motion

8th International Workshop on Robotics Software Engineering (RoSE'26), June 1st, 2026, Vienna, Austria

ykri@mmmi.sdu.dk (Y. Kristanto)

Sequences [2, 11]. This tight coupling means that even when similar programs exist in organizational repositories, identifying and adapting them remains difficult.While these statements are qualitative, they illustrate that ineffective search support remains a barrier to efficient program reuse. Furthermore, the user population further complicates search interface design, as manufacturing environments employ workers with diverse expertise, ranging from technicians with minimal programming experience to expert roboticists.

To tackle this challenge, a search tool that helps collaborative robot users of all expertise levels discover relevant collaborative robot programs for new implementations is proposed. We test our approach by studying users' search behavior for robot programs for a manufacturing process in which workpieces are loaded and unloaded onto a CNC machine by a robotic arm; this process is called machine tending. Machine tending involves common manipulation task requirements [3] under real-world hardware constraints, making it a suitable proxy for evaluating general manipulation systems. This work investigates how interface design and domain expertise interact to shape search behavior in robot program discovery. We address three research questions:

- **RQ1: Interface Effectiveness** How do wizard-guided filtering versus search-by-example approaches affect search success, efficiency, and subjective experience across expertise levels?
- **RQ2: Expertise and Search Strategies** How does domain expertise influence query formulation patterns, result evaluation behaviors, and criteria for assessing program adaptability?
- **RQ3: Configuration Constraints** How do users reason about hardware compatibility and adaptation requirements when evaluating program suitability?

2. Related Works

Despite progress in code search and software reuse, discovering and adapting robot programs across hardware configurations remains unsolved spanning robot program reuse, code retrieval, and expertise-driven information seeking, each offering only partial solutions to configuration-constrained program discovery.

2.1. Robot Program Reuse Challenges

Studies have documented challenges to the reuse of robot programs. First, search and discovery pose significant challenges. Estefo et al. [4] surveyed 182 Robot Operating System (ROS) developers and found that 74% failed to successfully reuse existing packages, with participants reporting that the search process itself was time-consuming and often fruitless. Developers struggled not only to locate potentially relevant packages but also to assess whether found programs could be adapted to their specific hardware configurations. Second, configuration dependencies create fundamental barriers to program portability across robot platforms. Biggs and MacDonald [2] demonstrated that programs are tightly coupled to specific hardware setups, with dependencies manifesting at multiple levels: kinematic parameters vary by robot model, motion constraints depend on workspace geometry and reachability, and control strategies differ based on sensor capabilities and end-effector characteristics. Stenmark and Malec [11] observed that robot programming knowledge often remains tacit and poorly documented, residing in the experience of individual practitioners rather than in accessible repositories. Without an explicit articulation of configuration requirements and adaptation strategies, programmers cannot systematically determine which existing programs might serve as suitable starting points, thereby forcing them to rely on informal knowledge networks or trial-and-error exploration. Robot program reuse constitutes a multi-dimensional search problem — demanding simultaneous matching across functional, physical, and hardware constraints absent from conventional code search tools, motivating specialized retrieval interfaces.

2.2. Code Search and Domain-Specific Retrieval

Research on code search in software engineering offers relevant techniques, but their applicability is limited to configuration-constrained domains. Traditional approaches employ keyword matching [1], structural similarity using abstract syntax trees [7], or neural embeddings [5]. Sadowski et al. [10] found that search success depends heavily on query formulation and domain knowledge, but their work, like most code search research, assumes programs are configuration-independent.

Clone detection methods [9] identify structurally similar code using techniques like longest common subsequence [6]. While useful for finding duplicate logic, these methods do not address compatibility constraints. As discussed in Section 2.1, the physical nature of robotic programs introduces a different form of complexity, making them difficult to reuse in ways that differ from general code. Two robot programs may be structurally identical yet incompatible if they assume different hardware configurations. Despite the documented need for robot program reuse support, no comparable specialized search tools exist in robotics.

2.3. Expertise and Information Seeking

Domain expertise fundamentally shapes information seeking strategies. White and Roth [13] synthesized research on exploratory search, showing that experts tend to formulate more precise queries, evaluate results more efficiently, and adapt their strategies more flexibly than novices. Wildemuth [14] further demonstrated that domain knowledge influences not only search success but also users' approach: experts leverage structural representations of a domain, while novices rely more on surface-level features and terminology.

Marchionini [8] distinguished between lookup tasks—locating known items—and learning tasks, which involve exploring to understand an unfamiliar domain. Novices are more likely to engage in exploratory behavior to build foundational understanding, whereas experts often pursue targeted retrieval. Vakkari [12] extended this work by showing that information needs and search tactics evolve alongside users' domain knowledge. However, prior work has not examined how domain expertise influences search behavior in configuration-constrained technical domains.

3. Search Tool Overview

We present a search tool for robot program discovery under heterogeneous configuration constraints, accommodating novice users seeking interpretable cues and experts navigating by structural program characteristics.

The design of the search tool is guided by two main goals. **First**, the system must support users with varied expertise levels by enabling access to reusable programs even when users cannot articulate queries in appropriate technical language. **Second**, the system supports two search strategies: top-down via task objectives and hardware configuration (Search by Wizard), and bottom-up via structural similarity to an example program (Search by Example). These goals align with exploratory search literature emphasizing flexible, iterative, and investigative interactions [13].

To support these goals, the system provides a wizard-based search mechanism. **Search by Wizard** implements a metadata-driven search process that guides users through a sequence of filtering actions. As users make selections, the wizard progressively narrows the candidate set. The configuration options are tailored to machine tending environments. Therefore: (1) robot model selection, (2) manipulated object properties, (3) gripper type and (4) optional constraint refinement (e.g., CNC door interaction). Alternatively, **Search by Example** enables users to specify their query by constructing action sequences. The queries are constructed by sequencing high-level actions displayed as actions. Sequences can be partial, specifying only distinctive actions. The system then compares the query with all programs in the database based on the sequence of actions. Each program is encoded as a sequence of actions (`Home`, `Move`, `Grip`, `Release`, etc). This ensures retrieval of behaviorally similar programs even when descriptive metadata is sparse or inconsistent — a recurring issue in robot programming.

4. Preliminary Evaluation

Building on the search tool design, we conducted a controlled user study investigating how interface design and domain expertise interact to shape robot program search behavior. The study addresses interface effectiveness (RQ1), expertise-dependent strategies (RQ2), and configuration constraint reasoning (RQ3).

4.1. Robot Program dataset, task complexity, and results filtering

The robot program dataset was derived from a replicated machine tending setup designed to reflect the key characteristics of a real workstation. The dataset comprises 8 programs organized into two complexity levels. The *first level* (Fig. 2a) comprises simple pick-and-place movements involving one or more objects without additional constraints only up to 5 statements. On the other hand, the *second level* (Fig. 2a) introduces industrial placement constraints, requiring specific object orientations and additional tasks (e.g., opening or closing a CNC machine door), reflecting more realistic conditions which may consists of 6 or more statements.

(a) First level complexity

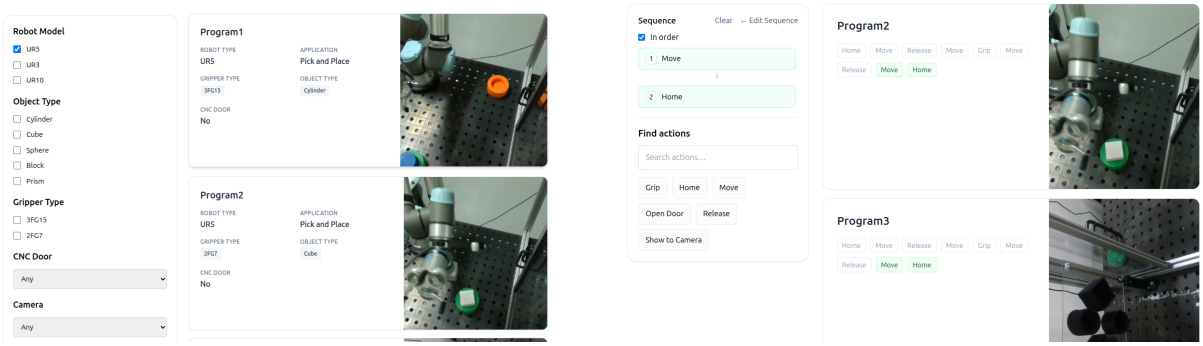
(b) Second level complexity

Figure 2: Scenario Complexity Variations

The filtering process for Search by Wizard is based on the Boolean retrieval method [15]. This method filters out possible robot programs by checking differences between the queried configuration parameters and the configuration parameters in the robot program's metadata. The Search by Example algorithm uses sequential graph matching to filter programs, verifying each action in order and returning only those that match the complete specified sequence.

4.2. Participants and Procedure

As one of our goal is to investigate how domain expertise influence their behavior in developing machine tending application, we evaluated our tool with both novice and expert users. Therefore we recruited 10 participants (5 novices, 5 experts, 3 females; mean age = 27.9) based on robot programming experience. Novices had no robotics experience. Experts had over a year of robotics experience developing robotic systems. In each task, participants are shown a video of a human-performed operation to be automated by a robot system, and are then asked to search for programs they can use as a base to realize it. Using a within-subjects design [17], each participant completed one search task per interface (Search by Wizard and Search by Example), with complexity balanced across both (see section 4.1). To reduce learning bias, task order is counterbalanced across participants. For instance, participant A completes the Search by Wizard task before the Search by Example task, whereas participant B follows the reverse sequence. To isolate each method's performance, the two search strategies were applied in separate tasks, with result interfaces (Fig. 3) showing only their respective query configurations, ensuring differences in search effectiveness are attributable to the method itself. A participant's program choice is considered correct when it requires minimal modification for use, though success criteria differ between tasks. In the Search by Wizard task, the chosen program must match the task requirements (e.g., handling the same object shape), while in the Search by Example task, it must serve as a viable base requiring fewer than 30% of its statements to be modified — as all shown operations are intentionally designed with this constraint in mind. Finally, the participant's perspectives of the tool and query formulation are captured through think-aloud [16] protocols and a short interview.



(a) Modified Search by Wizard Results Interface

(b) Modified Search by Example Results Interface

Figure 3: Modified search interface results for preliminary evaluation

5. Results

We analyzed data using mixed-effects models with expertise (novice, expert) as between-subjects factor and interface (wizard, Search by Example) as within-subjects factor. Given the exploratory sample size (n=10), we report effect sizes alongside significance tests.

Search Performance (RQ1): Experts achieved higher overall success rates (90%, 9/10 tasks) compared to novices (70%, 7/10 tasks), though this difference was not statistically significant, $t(18) = 1.10$, $p = .288$, $d = 0.49$. The medium effect size indicates a practically meaningful difference, though a larger sample would be needed to achieve stronger statistical power. Interface type influenced success rates across both groups. The wizard interface yielded 90% overall success (9/10) while Search by Example yielded 70% success (7/10). However, no expertise × interface interaction emerged: both experts and novices performed better with the wizard. Experts achieved 100% success (5/5) with the wizard compared to 80% (4/5) with Search by Example. Novices showed an identical pattern, achieving 80% success (4/5) with the wizard versus 60% (3/5) with Search by Example.

Search Strategies and Configuration Reasoning (RQ2, RQ3): Experts who succeeded with Search by Example (4/5) demonstrated the ability to formulate abstract action sequences matching task requirements. The one expert failure occurred when attempting to match a complex multi-object sequence. Novices who failed Search by Example struggled to construct appropriate abstraction levels for their queries, often specifying either overly generic or excessively detailed action sequences. With the wizard interface, experts achieved perfect success (5/5) by systematically specifying configuration constraints. Novices largely succeeded (4/5) with wizard guidance, with the single failure attributed to misunderstanding workspace constraint specifications during the filtering process. The wizard's structured approach appeared to compensate for novices' limited domain knowledge by making configuration dependencies explicit.

Subjective Ratings Post-task Likert scales (1-5) revealed nuanced patterns. For selection confidence, novices reported higher confidence with the wizard ($M = 3.60$, $SD = 0.55$) than Search by Example ($M = 2.80$, $SD = 1.30$), while experts showed the reverse pattern (wizard: $M = 3.00$, $SD = 1.41$; example: $M = 3.33$, $SD = 1.15$). Both groups rated tasks as moderately difficult (overall $M = 3.0$). Novices found the wizard easier ($M = 2.40$, $SD = 0.55$) than Search by Example ($M = 3.20$, $SD = 1.30$), consistent with their higher wizard success rates. The final interview with the participants revealed that Experts (5/5) preferred the use of Search by Example, and the Search by Wizard was preferred by novice users. On the other hand, Experts preferred the Search by Example method because this method allows experts to specify exact action sequence, giving them precise control to nuanced requirement that configuration parameters doesn't cover. This level of control was particularly valued by experts who understood the underlying robot mechanics and could formulate precise query. Conversely, novices preferred the Search by Wizard method because they were more comfortable specifying search queries through familiar configuration parameters. Furthermore, this approach provided structured guidance that reduced the cognitive burden of formulating queries from scratch.

6. Conclusion

Program reuse remains a significant challenge in robotic software development, hindering productivity and forcing users to recreate similar solutions repeatedly. To address this, we present a tool that enables users of all expertise levels to efficiently discover existing programs through two search methods: Search by Wizard, which filters programs using configuration-based queries, and Search by Example, which finds programs based on similarity to user-provided examples. Our study reveals that adaptive approaches — such as scaffolded guidance for novices and flexible pattern-based search for experts. While we recognize the limited sample size of our preliminary evaluation, future work will address this through larger-scale studies across diverse tasks and examine how users assess program relevance using a tailored comprehension tool [18].

7. Declaration on Generative AI

Claude and Harper were used for grammar, spelling, and vocabulary checks. All suggestions were reviewed and edited by the author(s), who take full responsibility for the content.

ACKNOWLEDGMENT

This work is supported by the Innovation Fund Denmark for the project FERA (3149-00014A)

References

[1] S. Bajracharya, J. Ossher, and C. Lopes, "Sourcerer: An infrastructure for large-scale collection and analysis of open-source code," Science of Computer Programming, vol. 79, pp. 241–259, 2014.

[2] G. Biggs and B. MacDonald, "A survey of robot programming systems," in Proc. Australasian Conf. Robotics and Automation, vol. 1, 2003, pp. 1–3.

[3] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," Robotics and Autonomous Systems, vol. 156, p. 104224, 2022.

[4] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry, "The robot operating system: Package reuse and community dynamics," Journal of Systems and Software, vol. 151, pp. 226–242, 2019.

[5] X. Gu, H. Zhang, and S. Kim, "Deep code search," in Proc. 40th Int. Conf. Software Engineering, 2018, pp. 933–944.

[6] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge, UK: Cambridge University Press, 1997.

[7] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in Proc. 29th Int. Conf. Software Engineering, 2007, pp. 96–105.

[8] G. Marchionini, Information Seeking in Electronic Environment, vol. 9, 1996.

[9] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Science of Computer Programming, vol. 74, no. 7, pp. 470–495, 2009.

[10] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: A case study," in Proc. 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 191–201.

[11] M. Stenmark and J. Malec, "Knowledge-based instruction of manipulation tasks for industrial robotics," Robotics and Computer-Integrated Manufacturing, vol. 33, pp. 56–67, 2015.

[12] P. Vakkari, "Searching as learning: A systematization based on literature," Journal of Information Science, vol. 42, pp. 7–18, Feb. 2016.

[13] R. W. White and R. A. Roth, Exploratory Search: Beyond the Query-Response Paradigm, no. 3. Morgan and Claypool Publishers, 2009.

[14] B. Wildemuth, "The effects of domain knowledge on search tactic formulation," JASIST, vol. 55, pp. 246–258, Feb. 2004.

[15] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval. New York: McGraw-Hill, 1983.

[16] C. Lewis, "Using the 'thinking-aloud' method in cognitive interface design," IBM Research Report RC 9265, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.

[17] Greenwald, A. G. (1976). Within-subjects designs: To use or not to use? *Psychological Bulletin*, *83*(2), 314–320.

[18] Y. Kristanto, M. Campusano, and M. Alipour, "FERAViz: A Visualization Tool for Robotic Program Interpretation for Future Reuse," in *IEEE Conference on Information Visualisation*, 2025.