CoFrame: A System for Training Novice Cobot Programmers

Andrew Schoen,[1] Nathan White,[1] Curt Henrichs,[1] Amanda Siebert-Evenstone,[2] David Shaffer,[2] and Bilge Mutlu[1]
[1] *Department of Computer Sciences,* [2] *Department of Educational Psychology*
University of Wisconsin–Madison, Madison, Wisconsin, USA
{schoen,cdhenrichs,bilge}@cs.wisc.edu, {ntwhite,alevenstone}@wisc.edu, dws@education.wisc.edu

***Abstract*—The introduction of collaborative robots (cobots) into the workplace has presented both opportunities and challenges for those seeking to utilize their functionality. Prior research has shown that despite the capabilities afforded by cobots, there is a disconnect between those capabilities and the applications that they currently are deployed in, partially due to a lack of effective cobot-focused instruction in the field. Experts who work successfully within this collaborative domain could offer insight into the considerations and process they use to more effectively capture this cobot capability. Using an analysis of expert insights in the collaborative interaction design space, we developed a set of *Expert Frames* based on these insights and integrated these *Expert Frames* into a new training and programming system that can be used to teach novice operators to think, program, and troubleshoot in ways that experts do. We present our system and case studies that demonstrate how *Expert Frames* provide novice users with the ability to analyze and learn from complex cobot application scenarios.**

***Index Terms*—robotics operator training; robot programming interfaces; collaborative robots; novice users; expert models**

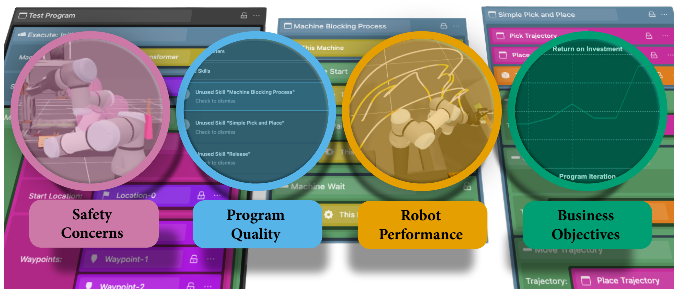

Fig. 1. In this paper, we present a new system, called *CoFrame*, that integrates a set of *Expert Frames* in collaborative robotics, focusing on *Safety Concerns*, *Program Quality*, *Robot Performance*, and *Business Objectives*, to train operators in using, programming, and troubleshooting cobot applications.

I. Introduction

More than a third of the facilities that use robotic technology employ collaborative robots (cobots) [1] and cobots deployed within the manufacturing context are expected to continue to grow in market share. Industry seeks to mitigate labor shortages [2] while improving workcell performance and reducing human worker's health risks. To this end, a new generation of manufacturing robots designed to work in a shared space alongside human operators as collaborators are replacing conventional caged robots. Work traditionally done by a human can be parceled into tasks that consider the skill-sets uniquely brought by humans and cobots [3]. Although much research and engineering effort has been done to bring these robots into the workcell, the training-procedures, tools, and practices to support human operators have lagged behind [4]. This lag results in a "skills gap" for operators working alongside cobots without the knowledge and skills to customize the robot's behavior to better accomplish the task [5].

Research has identified specific occupations such as craft work, where the skills gap in utilizing robotic technology is most pronounced [6], and how individuals differ in their specific skills and preferences regarding the use of cobots [7]. Other research has sought to better understand this skills gap, specifically to address the question, "what do workers need to know in order to effectively utilize these systems?" Siebert-Evenstone et al. [8] interviewed experts in collaborative robotics—including engineers, implementers, and trainers—to identify the skills, tools, and perspectives they utilize in troubleshooting and programming cobots, developing an "expert model" of collaborative robotics. Such models serve as opportunities to develop training, programming, and control interfaces for HRI research. In order to address the skills gap in collaborative robotics, we developed a digital training environment called *CoFrame* that aims to prepare traditional and non-traditional students as operators of cobots. In this paper, we present the model of expert skills and knowledge that serves as the basis of our automated expert feedback system illustrated through several system capability case studies.

The contributions of our work include:

- An operationalization of the *Safety First* expert model as *Expert Frames* to address the cobot skills gap;
- Design and implementation of a design-learning environment that incorporates the *Expert Frames* [1];
- A set of case studies demonstrating system behavior and how it provides feedback during authoring/learning.

II. Related Work

In this section, we review related work on recent developments in collaborative robotics, the skills gap that results

[1]Code available at https://github.com/Wisc-HCI/CoFrame

from the introduction of these technologies into workplaces, interfaces for cobot authoring and programming, and the state of the art in cobot training systems.

A. The Emerging Field of Industrial Collaborative Robotics

Since their introduction to the market in late 2000s, cobots have found widespread adoption across industries, including manufacturing [9], logistics [10], and medicine [11]. Research in the last decade has investigated the safety and ergonomics of the use of cobots [12–14], how work can be structured to enable humans and robots to work together [3, 15], and how cobots can be integrated into production lines [16, 17]. Key insights for the success of cobots from this body of work include the promise of cobots to improve both the efficiency and ergonomics of some manual processes [3] (mostly in medium-sized production volumes [18]); the need for establishing well-defined levels of collaboration [15, 19] and forms of task interdependence [20]; and the importance of employee-centered factors such as the fear of job loss and ensuring an appropriate level of trust in the robot [21]. Overall, this is a rapidly emerging field involving the development of new technology to enable the integration of robots into work environments; study their safety, ergonomics, and effectiveness; and work toward understanding of how they affect human workers.

B. The Emerging Worker Skills Gap

Cobots offer many benefits across several industries, including productivity benefits to organizations and health and safety benefits to human workers, but the introduction of advanced technologies, particularly technologies involving automation, robotics, and advanced interfaces, into workplaces is creating a "skills gap"—a gap between the skills necessary to utilize these technologies and the skills of the existing workforce [4, 5, 22]. In the U.S., nearly half of the job openings, totaling 2.2 million positions, in manufacturing remain open due to a shortage of workers with the skills necessary to effectively utilize such technologies [23]. In the context of robotic technologies, the skills gap exists at all levels, from robot operators to researchers [24]. Although education and training have been proposed as the primary means of closing this gap [25], a recent analysis of existing educational programs found a lack of emphasis on critical technical and non-technical, or "soft," skills in these programs [26]. Despite showing that industry lacks the appropriate means, including curricula, materials, and knowledge, to offer such training, this analysis also highlights the importance of work-based, hands-on training and apprenticeships. Our work aims to capitalize on this promise by creating a training system that situates the learning in a real-world or simulated work environment.

C. Robot Programming Tools and Environments

An area of collaborative robotics where the skills gap is significant is the programming of cobots for new tasks [27]. Existing approaches to addressing this gap primarily involve the development of intuitive and ease-to-use robot programming that borrow ideas and concepts from end-user programming, such as the RoboFlow and Code3 visual programming languages [28, 29], and the application of these approaches to the programming of industrial robots [30]. Evaluations of the effectiveness of these approaches to enable adult novices to program cobot applications show them to be more effective, usable, learnable, and satisfactory compared to the existing cobot programming interfaces [31]. Research into end-user programming tools also include highly advanced robot programming tools that enable semantic skill demonstrations [32, 33], task allocations to human-robot teams [34], and AR-based interfaces that leverage workspaces as augmented programming surfaces [35–38]. However, these systems target intuitive and rapid programming of robots and do not address the skills gap by advancing the skills of the user in programming and troubleshooting cobot applications.

D. Robotics Training Systems and Programs

Prior approaches to addressing the skills gap in robotics highlighted the unique challenges of working with robotic systems, including manipulating real-world entities using software programs and situating these skills into real-world problems. For example, Dagdilelis et al. [39] developed a program that integrated visual programming to teach robot programming concepts to high-school students. Cobot systems have also been explored as a medium to teach students at the college level engineering design [40]. To address the challenge of situating learned skills in real-world problems through hands-on learning, prior work has proposed the concept of a "Teaching Factory" that offers a factory-like classroom environment [41, 42]. These environments offer trainees genuine systems, constraints, and problems to work on and opportunities to interact with both instructors and practitioners. Prior research in situating learning in genuine environments also includes the development of virtual- and augmented-reality based learning environments that enable trainees to perform work tasks and processes [43], although these systems aim to train workers in collaborating with robots rather than providing the skills necessary to program and troubleshoot them. Cobot manufacturers provide training programs targeting specific skills necessary to utilize their products, e.g., Universal Robots Academy [44]. These programs are used in vocational training [45], although effectiveness of these resources to address the skills gap is unknown, and experiences of early adopters of cobot systems indicate that they are not sufficient [4]. Some research has explored methods for translating expert knowledge to robot operators [46], although these methods have not been applied in training systems.

We previously discussed opportunities and challenges in collaborative robotics, particularly the need to address the skills gap that has become a bottleneck in the widespread adoption and utilization of cobots. Although the growing body of research in end-user programming tools can make it easier for workers to use cobots, addressing the skills gap requires new training programs and technologies that can help workers obtain expert problem-solving skills and apply them in real-world settings. Our training system, *CoFrame*, aims to address this need for cobot operators.

III. Expert Model

A. *Collaborative Expert Model*

Our implementation of the *Expert Model* relies on the findings of an ethnographic study by Siebert-Evenstone et al. [8] regarding how experts think about cobot application design. They found that expert thinking falls into a *Safety* + structure. Specifically experts keep *Safety* concerns (collaborative/shared space collisions, pinch-points, risk-assessment, force sensing, tool/part manipulation) in mind while considering other aspects of the program, such as *Performance Objectives* (cycle-time, speed, payload), *Business Objectives* (robot wear-and-tear, cost, ROI, efficiency), and the *Application* (problem-solving, flexibility/adaptability, robot reach, human interaction, positioning).

Experts bring a deep systematic understanding to their application design to balance a variety of critical safety points with concerns of cost and flexibility, and usability. They examine the use case to determine if a cobot is preferable over a traditional robot, which specific robot(s) to deploy, what sensors and integrations are needed, the process sturcture, and how that impacts safety. Traditional manufacturing robots are either physically caged or use sensors to detect entry into an exclusion zone, but cobots are designed to safely work around the human operator with appropriate programming, provided the integrated end-effector tooling and workspace are also safe.

In designing the application, experts weigh the cost of the engineering challenge for a robot to manufacture the part versus having a human operator perform the activity. Experts typically deploy operators in low interaction roles such as setup, starting robot, inspection, and ending the process; and emphasize that they should avoid getting in the way of the robot. Experts consider where the human operator is within the workspace, their role, and how they are trained to perform it. They design their applications to make use of external sensors (vision systems) and tools (conveyors, CNC mills). When an error occurs they have knowledge of what it means and are able to reason about appropriate solutions.

To translate the *Expert Model* developed by Siebert-Everstone and colleagues [8] into a form compatible with learning outcomes, we reorganized these concepts into four *Expert Frames*: *Safety Concerns*, *Program Quality*, *Robot Performance*, and *Business Objectives*. This mapping can be seen in Figure 2. Each of these frames represent a lens with which to assess the robot collaboration.

B. *Expert Frames*

Safety Concerns: This frame is heavily based on the *safety* theme from the *Expert Model* but incorporates aspects of the *operator* and *trajectory* themes. The *Expert Model* focuses on safety, going so far as to place it above the others in terms of importance. One expert indicated that “I could buy a collaborative robot, but if I’m moving around steak knives, it’s no longer collaborative, so there’s no point to using a collaborative robot” [8]. Thus, the collaboration incorporates

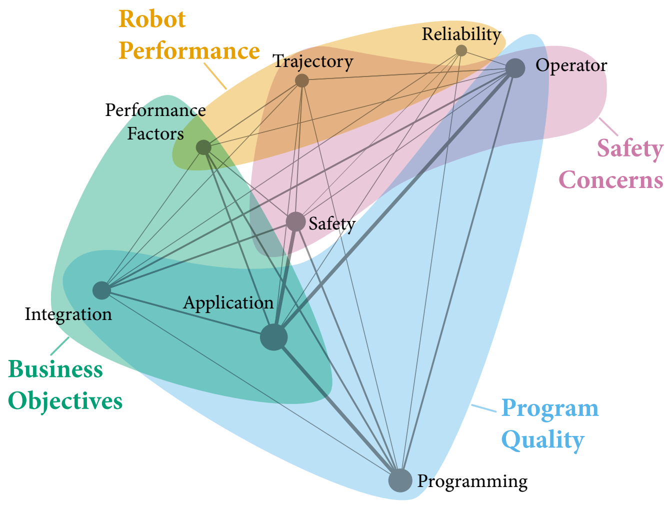

Fig. 2. The mapping of the themes from the *Expert Model* [8] into each of the four *Expert Frames*: *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (yellow), and *Business Objectives* (green). Figure adapted from Siebert-Evenstone et al. [8].

the orientation and safety of the robot’s tool, the safety of the individual objects that the robot may be carrying, whether the robot’s trajectories include possible pinch points or collisions, the robot’s space usage during the program, and how that interacts with the human’ collaborator’s space. Key to this frame is providing clear and concrete feedback about the safety of the resulting program; as individuals with less cobot programming experience may not design a task to be safe, unaware that they are violating certain safety heuristics [8].

Program Quality: Several themes from the *Expert Model*, including *programming*, *integration*, *application*, and *operator* combine to create this crucial and practical feedback frame. Many other frames depend on proper specification of the program to provide meaningful feedback. This frame includes simple program attributes, such as the parameter satisfaction, and more complex ones, such as how the robot must adapt to the duration of various machine processes. Specifically, we evaluate the program based on missing parameters and code blocks, unused skills and features, empty code blocks, and any logical issues regarding integration with machines.

Robot Performance: With a focus on robot execution quality and ability to perform actions, this frame includes aspects of *performance factors*, *reliability*, and *trajectory* from the *Expert Model*. In many cases, these performance metrics relate to the other frames, especially *Safety Concerns* and *Business Objectives*, and include qualities such as reachability, the speed of the joints and end-effector tool, payload, and space use.

Business Objectives: These outcome-oriented feedback metrics guide the design by enabling operators to consider how their changes to the program affect the profitability of the robot, or how wear-and-tear might be impacted. This frame is informed by the *performance factors*, *application*, and *integration* themes from the *Expert Model*, and focuses

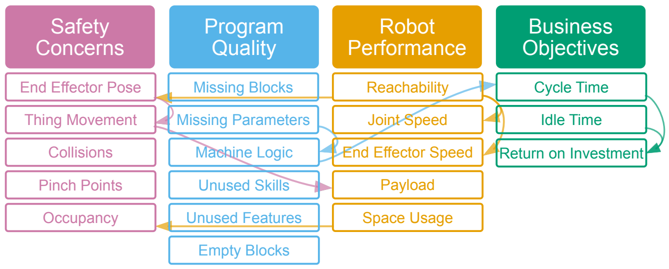

Fig. 3. The four *Expert Frames* of *CoFrame*, and the relationships between them. As operators address concerns in each frame, they unlock other considerations. For example, only after addressing whether a Location or Waypoint is reachable (*Robot Performance*), do they address issues with the pose of the end effector.

specifically on cycle and idle times of the robot, and the return on investment.

C. Frame Relationships

In addition to specifying themes commonly discussed by cobot experts, the *Expert Model* describes the relationships among them. As already noted, expertise is usually identified not so much by the knowledge of isolated facts or heuristics, rather by a deep and complex understanding of the relationships between them within a domain [47–49]. We operationalized these relationships by linking the individual sections for frames to other sections within or outside the parent frame. In some cases, these section relationships are practical. For example, a coherently defined program with regards to machine logic (*Program Quality*) enables calculation of total cycle time (*Business Objectives*). In other cases, they reflect the logical sequences of expert concerns. For example, determining whether a certain robot pose could introduce pinch points (*Safety Concerns*) is only relevant after considering whether the robot can reach the pose (*Robot Performance*).

In our *Expert Frames*, we use these dependencies and relationships to guide the operator to work through the logical dependencies whilst developing the connections and associations between them. A full list of dependencies between the sections of each of the frames can be seen in Figure 3.

IV. System Design & Implementation

We developed *CoFrame* using the *React* framework that communicates with a backend running a PyBullet [50] simulation. Visualizations are rendered in *Three.js*. Our system is designed around four "tiles" as shown in Figure 4: the Program Editor tile (G), the Simulation tile (B), the Contextual Information tile (C), and the Expert Frames tile (A).

A. Programs and Program Editor

We implemented a block-based visual programming language for operators to build their programs, heavily inspired by other commonly used tools like Blockly [51] and Scratch [52].

1) Code Block Design: We utilized a drag-and-drop mechanism that allows an operator to easily construct viable programs (Fig. 4 F) and visualize the connections between different blocks. This is accomplished by dragging blocks from the block drawer (Fig. 4 D) into the canvas (Fig. 4 E). Each block type is given a distinct color and icon in order to assist in this visualization process. As operators build their program, they can highlight blocks in the program editor to receive more information in the Contextual Information tile, as well as visualize the action in the Simulation tile if fully specified.

2) Block Types: There are two main categories of blocks—item and executable. Item blocks refer to objects in the workspace (e.g., things, trajectories, machines, locations, and waypoints). These item blocks are used to parameterize the executable blocks, such as actions like "Move Gripper" (accepts the *thing* being gripped or released by the tool), "Move Trajectory" (accepts a trajectory item block), or "Machine Start" (accepts the machine to start). Some actions take additional numerical or configuration parameters, such as movement speed or gripper position (e.g., in the "Move Gripper" action).

Machines are item blocks that create and modify *things* (parts and materials the robot interacts with) within the program. In our simulation, these machines are the 3D printer, conveyors, and the assembly jig. They specify recipes with inputs, outputs, and processing times. Things are objects that can be produced as output of machines, consumed as inputs, and moved by the robot. They also specify various properties, such as weight and safety (*e.g.*, a blade is unsafe to carry unsheathed).

Locations and waypoints are positions in 3D space that represent where the user may want the end effector to be, both in terms of translation and rotation. Locations are presented to the operator as places where the robot would start and stop its movements, such as the end of the conveyor machines or the assembly jig, and are used in trajectories. Waypoints are used to specify intermediate positions between locations, usually to

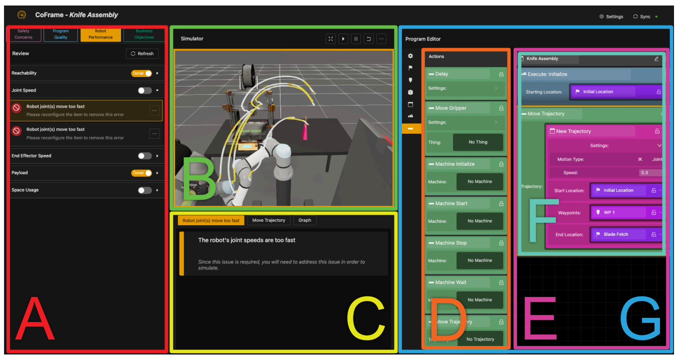

Fig. 4. The layout of the *CoFrame* interface. Operators can use the Program Editor tile (G) to construct their program, and can visualize the results in the Simulator tile (B). The Expert Frames tile (A) allows them to swap between different *Expert Frames* and view issues in each frame. When not viewing issues, the Contextual Information tile (C) shows relevant frame-related information, and when viewing issues also provides detailed information about the issue and suggestions for changes. Within the Program Editor (G) operators can drag blocks from the Block Drawer (D) into the Program Canvas (E). The Program Canvas contains the program (F) along with implemented skills.

guide the robot to perform more desirable motions. Trajectory blocks accept parameter item blocks (locations and waypoints). They represent the motion the robot will execute; beginning at the start location, navigating through the an ordered list of waypoints, and stopping at the end location. They are also parameterized with movement speed and interpolation type (inverse-kinematic or joint-based).

Executable blocks within *CoFrame* fall into three categories: actions, groups, or skills. Actions, the smallest indivisible units of code, accept various parameters specifying their behavior. Actions that affect robot state can be simulated when selected in the program editor and fully parameterized.

The “Delay” action stalls the program for an adjustable amount of time, allowing the operator to explicitly specify a time that the robot is inactive, e.g., to allow the human to perform a task. Similarly, “Breakpoint” actions are a debugging action that stops the execution of the program at that block.

The “Machine Initialize, -Start, -Stop, and -Wait” actions all take in a *machine* parameter. “Machine Initialize” is equivalent to turning on the machine, and needs to be performed before any other machine actions. “Machine Start” begins the associated recipe with any matching *things* required. “Machine Stop” indicates that a machine has completed but cannot be executed until after processing from the paired “Machine Start” ends. When completed, the output *things* are available to be interacted with. “Machine Wait” allows the operator to specify that the robot waits for whatever time remains on a machine process and guarantees that the “Machine Stop” does not occur before the minimum processing time of the machine has completed.

Regarding robot control, operators have access to the “Move Gripper”, “Move Trajectory”, and “Move Unplanned” actions. “Move Gripper” allows the user to manipulate the gripper to either grasp or release a specified target *thing*. It takes additional parameters for the desired final distance between the gripper fingers, and its movement speed. “Move Trajectory” takes in a *trajectory* parameter for the robot to execute, and moves the robot along the specified motion. “Move Unplanned” takes in a *location* parameter and is a way for the operator to specify starting locations or human-driven operation.

“Group” blocks are action blocks that function as a container for other actions, allowing the operator to group actions into coherent code blocks. These can be collapsed to reduce visual clutter and be previewed in the simulator when fully specified.

“Skill” blocks behave similar to functions in other programming languages, defining a set of parameters, a context of use for them, and action blocks that are executed. “Skill Call” blocks are generated for each “Skill” block in the operator’s program. Like other action blocks, these blocks accept item parameters and pass them along to the paired “Skill” blocks.

B. Simulation

The Simulation tile visualizes the robot; its movements, actions, the environment; and frame-based or issue-based feedback. At the onset, the simulation visualizes the robot going through the execution of the program. The simulation also connects with the Expert Frame tile to provide any extra visual information while showing the relevant robot animation.

C. Contextual Information

The Contextual Information tile provides operators with frame-specific feedback and suggestions about selected blocks and items. The displayed information changes based on what they select and interact with in the Simulation, the Program Editor, and the Expert Frame tile. This includes definitions for terms and phrases novice operators are exposed to, frame dependant information, as well as various graphs for selected issues. Definitions are provided for words that are commonly used by experts and within robotics programs as well as explanations for how they relate to other terms.

Based on the selected frame, the Contextual Information tile provides additional prompts to help the user think about the concepts within each frame. For example, when adding waypoints to trajectories with the *Safety Concerns* frame selected, it will show, “Pay special attention to placing waypoints around the occupancy zone of the human, since this is more likely to result in undesirable conflicts between the human and the robot.”

D. Expert Frames

Prior work highlighted the need for programming environments to provide users with a comprehensive list of *issues* and to automatically collect and display information about program execution [53]. The design of the Expert Frame tile (Fig. 4 A) builds on these guidelines, automatically providing feedback about the operator’s program through *issues*. Each issue represents a unit of feedback regarding an expert concern for a particular element of the program, and is complimented with textual or visual data to give information about how the program was executed. For example, when viewing issues about pinch points, the corresponding “Move Trajectory” block in the program editor is highlighted, and the operator is presented a simulation view of the robot moving through a trajectory that highlights the pinch points to draw attention to the issue.

Issues are marked as either warnings or errors. When marked as warnings, they are displayed in the Expert Frames tile with gray icons. When marked as an error, they are displayed in red. Warnings and errors also differ functionally, as warnings can be manually marked as fixed, allowing the operator to address other issues in the Expert Frames tile, while errors require the operator to make adjustments to their program.

1) Safety Concerns: End effector pose issues refer to cases where the gripper moves quickly in the direction of its fingers. Each trajectory timestep is scored and shown as a graph in the Contextual Information tile when the issue is selected. The simulation displays an animation of the robot moving through the trajectory along a line marking gripper trajectory, which is colored to visually indicate the ratings of the different portions of the trajectory. Operators are first required to address any *reachability* issues before addressing end effector pose.

Thing movement issues represent cases where the robot moves potentially unsafe objects through the space. While progressing through the program, checks are made for whether gripping the *thing* is possible given the orientations of both the gripper and *thing*. If in the program the robot executes a “Move Trajectory” action whilst carrying an unsafe object, it is flagged with an error and visualized in the 3D scene.

Each valid trajectory is analyzed with PyBullet collision detection [50] to detect potential pinch points. These are visualized as moving dynamic spheres placed around the robot as it moves along the trajectory, such that larger, darker spheres are higher priority.

Collision issues occur when a trajectory causes the robot to collide either with itself or the environment. When selected, the corresponding trajectory is highlighted in the program editor, a graph depicting the robots proximity to itself and the closest object in the environment, and the simulation displays an animation of the robot moving through the trajectory along with lines marking the path each of the robot’s linkages take. Darker colors along the lines indicate closer proximity.

Occupancy issues refer to instances where a robot trajectory overlaps with the human occupancy zone and share the same visual cues as collisions issues. Occupancy issues have a dependency on *space usage*, as the higher space usage correlates with an increased likelihood of entering the occupancy zones.

2) Program Quality: Missing block issues identify cases where the operator has failed to supply a necessary block where needed, such as in a “Move Trajectory” action. Similarly, missing parameters refers to instances where the code block does not have all the required parameters.

Machine logic issues identify where the program specifies invalid interactions with machines, for example when a machine is stopped before it finishes processing. Before an operator can address machine logic issues, they must first address any missing parameters.

Unused features help operators identify cases of unused item blocks. Similarly, unused skills refer to operator-defined skills that are not called in the executed program. When these types of issue are selected, the corresponding unused block is highlighted in the program editor.

Empty blocks refer to instances where either the program, a group, or a skill does not contain any actions. Selecting this type of issue highlights the empty block in the program editor.

3) Robot Performance: Reachability issues refers to instances where the robot is not able to move to a given waypoint or location because a solution cannot be found, or the position is out of the robot’s reach. When selected, the corresponding waypoint or location is highlighted in both the simulation and program editor, and a window opens allowing the operator to adjust the waypoint or location’s position in the environment.

Joint speed issues are instances where the robot’s joints exceeding a threshold value for a trajectory. When selected, the corresponding trajectory in the program will be highlighted, a graph of each joint’s speed over time is displayed in the Contextual Information tile, and an animation of the robot executing the trajectory with lines for each of the joints colored by their speed is shown in the simulation. Before operators are able to address these issues, they are required to first fix any issues with *reachability*, as the execution of trajectories is dependent on reaching the locations and waypoints along the way. End effector issues function similarly to joint speed, instead showing the speed of the end effector. When selected, it shows similar visuals to joint speed issues, with a graph and animation of the end effector. For similar reasons to joint speed, operators must first fix any issues with *reachability*.

Payload issues occur when the robot lifts *things* that approach or exceed its carrying capacity. If such a violation occurs in a ”Move Trajectory,” it is highlighted in the program editor and simulation. Payload issues require operators to first address issues with *thing movement* to fix other non-safe manipulations.

Space usage issues refers to the percentage of the robot’s workspace utilized at any point during a given trajectory. When selected, the program editor highlights the corresponding trajectory; the Contextual Information tile displays a graph of how the utilization changes over the course of the trajectory; and the simulation shows a convex hull of the trajectory.

4) Business Objectives: Cycle and idle time issues are always generated and are encoded as persistent warnings. Cycle time refers to the total time that it takes for the robot to complete the operator’s program once, while idle time refers to just the amount of time a robot is spent idling during program execution. Similarly, return on investment (ROI) issues are always generated and refer to the ratio of product value to total cost building the product, including the cost of robot wear-and-tear - which is based on the robot’s acceleration. When these issues are selected, the Contextual Information tile displays a graph showing how the time changes as the operator adjusts their program. ROI issues require operators to first address both cycle and idle times issues, as ROI is dependent on them.

V. Case Studies

To demonstrate the behavior of *CoFrame* as a learning-programming environment, we developed three case studies that illustrate how it detects and responds to issues generated by the operator to support learning. To accomplish this, we specifically designed a task to prompt certain issues that the operator will have to address. The task is based on an expert’s comments [8] and has the robot assembling a knife from a set of components, namely a blade and two halves of the handle. The knife itself is unsafe to carry and the system will notify the operator if attempted. They will instead have to use a safety transport jig that covers the blade making it safe to carry. Blades arrive from a conveyor, handles are produced with a 3D printer, parts are assembled in a jig, and the finished knife is deposited on another conveyor. Figure 5 depicts each of the case studies detailed below.

A. Case Study 1: Defining a trajectory

One of the first substantive actions an operator will attempt is creating a trajectory to move the robot. They would likely start by considering the 3D scene and inspecting each machine to observe where the robot needs to move first. They will see that they need to move to the blade receiver, which catches the blades as they arrive from the conveyor. Then

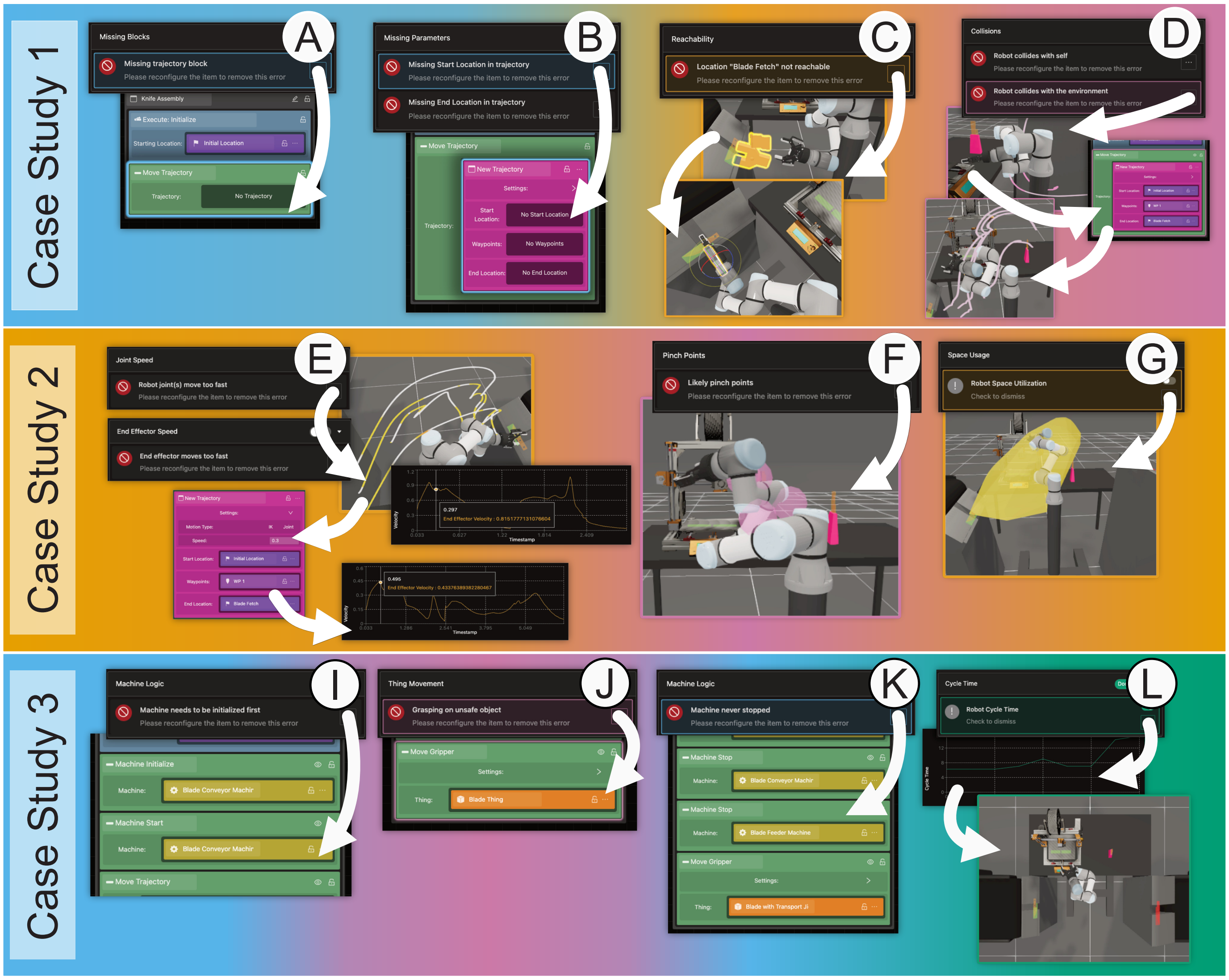

Fig. 5. Three case studies showing the process of evaluating feedback from the system and informing adjustments to the operator's program. The gradient background of the figure denotes the switching between *Expert Frames* by the operator, from *Safety Concerns* (pink), *Program Quality* (blue), *Robot Performance* (orange), and *Business Objectives* (green). In Case Study 1, the operator begins by addressing a missing trajectory block (A), followed by filling in its parameters (B). The operator then addresses reachability concerns (C). They finish by addressing issues with robot collision (D). In Case Study 2, the operator begins by addressing joint speed issues and visualizes the speed (E). They transition to solving pinch point issues (F). They finish by addressing issues with the robot's space usage (H). In Case Study 3, the operator begins by solving issues with uninitialized machine logic (I), then addressing problems with thing movement (J). They return to addressing a machine logic for a non-stopped machine (K). The operator finishes by viewing the robot's cycle time (L).

they open the program editor's block drawer and drag a "Move Trajectory" block into the program. They click on the "Refresh" button in the Expert Frames tile refreshing the program, showing a number of errors. They realize the action requires a "Trajectory" block. They refresh the feedback and are prompted to parameterize the trajectory with a start and end location. Unfortunately their end location is unreachable. They click on the issue to bring up the location for editing and notice the robot got stuck in a joint state preventing it from reaching the location. The operator adjusts the location. *CoFrame* then displays the new joint state that aligns the gripper with the location. At this point, the operator refreshes. They see the trajectory has been fully specified. However, the trajectory causes collisions with both the environment and itself. They add a waypoint, guiding the robot above the table and avoid colliding with itself. After a last refresh, the collision issues have been downgraded to warnings.

B. Case Study 2: Debugging a movement

After specifying a syntactically valid trajectory, an operator may want to evaluate its performance. They click the *Robot Performance* frame showing active issues on joint and end effector speeds. Clicking the joint speed issue plots lines for each joint position through time in the 3D scene. The operator also clicks the end effector issue and observes the graph in the information section. The operator tweaks the speed parameter to resolve the issue. They switch back to the *Safety* frame to address pinch point violations. The operator adds a waypoint to better coax the robot to a joint state that prevents the issue. After adding a waypoint, they click the feedback "Refresh" button to update the trajectory visualization. The operator is now prompted to address robot space usage for the trajectory. They notice that the robot extends out into the workspace more than necessary so they again tweak the waypoints; iterating over speed, collision, and pinch point concerns.

C. Case Study 3: Working with machines

The operator revisits the scene to consider the machines' operations. They click on the blade conveyor and see that it "produces blades," which they want to move to the assembly jig. They open the block drawer and place a "Machine Start" action after their trajectory; parameterizing the action with the machine's item block. They then place a "Machine Wait" action. Refreshing the feedback, they see an error that the machine needs to be initialized before use. The operator adds the necessary action block, then adds a "Move Gripper" action parameterized with the blade, followed by a second "Move Trajectory" action. They iterate over the new trajectory in a similar fashion to the first one, though they add and adjust waypoints before seeking frame feedback.

After refreshing the feedback, they encounter a thing movement issue called "Grasping unsafe object." Realizing the issue is with the blade being grasped, the operator focuses on the simulation to find the blade receiver machine, which converts a blade and a transport jig into a safe blade with transport jig *thing*. They then add a new set of machine actions (initialize, start, and wait) to the program. Refreshing feedback, they find an error "Machine never stopped" for both the conveyor and receiver. They add the necessary actions and inspect program operation. Curious about the *Business Objectives* frame, they toggle the frame and inspect the cycle time. The operator views the graph in the Contextual Information tile, and decides to find a more optimal program. They tweak the order of the blade conveyor "Machine Start" to happen before moving the robot to the blade receiver, reducing cycle time.

VI. Discussion

Multiple industries currently face a skills gap in effectively utilizing cobots in the workplace. Designing cobot applications requires considerable expertise that few workers currently have, presenting difficulties for companies looking to incorporate cobots alongside their human workers. Critically, workers generally lack the expertise to effectively construct, adapt, and debug cobot programs. However, this situation also presents opportunities for job creation if effective instruction methods can be created which narrow this skills gap.

To better understand what content these methods must communicate and teach, research by Siebert-Evenstone et al. [8] identified a set of themes that form a *Safety First Expert Model* of cobot expertise. We translated this model into a set of *Expert Frames* that can be used to instruct novice cobot programmers in the content and relationships of the *Expert Model*. Next, we presented an implementation of a combined learning-designing environment that provides interactive textual and visual feedback for operators' programs in accordance with the *Expert Frames*. Finally, we provided a set of case studies that illustrate the pathways through these *Expert Frames* that operators may trace, thereby creating and reinforcing associations of content within the *Expert Model*.

The process of performing this translation and development was informative as well. While the majority of relationships between the content of each frames is derived from the *Expert Model*, a number of others arose naturally from the design of the system and the requirements of providing feedback. For example, a number of *Program Quality* attributes (e.g. full parameterization) are required as a necessity of deriving higher-level feedback on things like *Machine Logic* and *Cycle Time*. Furthermore, designing frame-based feedback at multiple levels of detail and at various levels of program completion requires clear, concrete, and data-driven outcome measures.

A. Limitations & Future Work

We have not yet conducted a planned empirical evaluation of *CoFrame* with experts to assess the translation of the *Expert Model* into *Expert Frames* within *CoFrame*. Although our case studies illustrate system behavior and capabilities, future work will complement this demonstration with an efficacy study with novices comparing *CoFrame* to other methods.

Since program traces can be visualized both in isolation or within the complete program, this presents a challenge in matching the joint states of consecutive trajectory movements. A more reliable method that incorporates a better model of all possible program traces could alleviate any errors that do arise. The current version of our system does not address issues like slip and uncertainty related to gripping real-world objects, nor does it integrate with physical robot systems. Both may be considered in future work. *CoFrame* was built around an *Expert Model* that drew on level-one (e.g., start-stop shared-space, time-separated) collaborations commonly found in industry [4]. Future work should consider more dynamic, reactive human awareness and explicit modeling of human-robot interaction within the program. *CoFrame* is not currently designed to be customized for alternative setups or to handle real-time control, but integrating these features may allow it to be more general-purpose and offer unique feedback opportunities. Given the importance of visual, contextual feedback in the system, we plan to consider including Mixed Reality.

CoFrame represents a novel translation of expert-derived knowledge regarding the design of cobot programs, and a unique opportunity for future research regarding the efficacy of such tools as methods for narrowing the skills gaps present across industries.

VII. Acknowledgements

This work was supported by the National Science Foundation under awards IIS-1822872 and IIS-1925043. We would like to thank ZeDong Zhang and Yiran Zhou for their assistance during the development process.

References

[1] D. Miller, *Robotics Adoption Survey Finds Ups, Downs, and a Few Surprises*, Online; accessed August 30, 2021, Mar. 2021. [Online]. Available: https://www.automationworld.com/factory/robotics/article/21307161/robotics-adoption-survey-finds-ups-downs-and-a-few-surprises.

[2] D. Autor, "Good news: There's a labor shortage," *The New York Times*, 2021. [Online]. Available: https://www.nytimes.com/2021/09/04/opinion/labor-shortage-biden-covid.html.

[3] M. Pearce, B. Mutlu, J. Shah, and R. Radwin, "Optimizing makespan and ergonomics in integrating collaborative robots into manufacturing processes," *IEEE transactions on automation science and engineering*, vol. 15, no. 4, pp. 1772–1784, 2018.

[4] J. E. Michaelis, A. Siebert-Evenstone, D. W. Shaffer, and B. Mutlu, "Collaborative or simply uncaged? understanding human-cobot interactions in automation," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[5] J. Wingard and C. Farrugia, *The Great Skills Gap: Optimizing Talent for the Future of Work*. Stanford University Press, 2021.

[6] J. R. Holm, E. Lorenz, and J. Stamhus, "The impact of robots and ai/ml on skills and work organisation," in *Globalisation, New and Emerging Technologies, and Sustainable Development*, Routledge, 2021, pp. 149–168.

[7] G. Giannopoulou, E.-M. Borrelli, and F. McMaster, "" programming-it's not for normal people": A qualitative study on user-empowering interfaces for programming collaborative robots," in *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, IEEE, 2021, pp. 37–44.

[8] A. Siebert-Evenstone, J. E. Michaelis, D. W. Shaffer, and B. Mutlu, "Safety first: Developing a model of expertise in collaborative robotics," in *International Conference on Quantitative Ethnography*, Springer, 2021, pp. 304–318.

[9] A. C. Simões, A. L. Soares, and A. C. Barros, "Factors influencing the intention of managers to adopt collaborative robots (cobots) in manufacturing organizations," *Journal of Engineering and Technology Management*, vol. 57, p. 101 574, 2020.

[10] I. Lappalainen, "Logistics robots as an enabler of hospital service system renewal?" In *The 10 years Naples Forum on Service. Service Dominant Logic, Network and Systems Theory and Service Science: Integrating three Perspectives for a New Service Agenda. Ischia, Italy*, 2019.

[11] J. Ernst and C. Jonasson, "Serving robots?–exploring human and robot social dynamics in everyday hospital work," in *36th EGOS Colloquium 2020: Organizing for a Sustainable Future: Responsibility, Renewal & Resistance*, 2020.

[12] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg, and R. Mellander, "Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept," in *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, Ieee, 2011, pp. 1–6.

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

[14] L. Gualtieri, E. Rauch, and R. Vidoni, "Emerging research fields in safety and ergonomics in industrial collaborative robotics: A systematic literature review," *Robotics and Computer-Integrated Manufacturing*, vol. 67, p. 101 998, 2021.

[15] J. Shi, G. Jimmerson, T. Pearson, and R. Menassa, "Levels of human and robot collaboration for automotive manufacturing," in *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, 2012, pp. 95–100.

[16] M. Wojtynek, J. J. Steil, and S. Wrede, "Plug, plan and produce as enabler for easy workcell setup and collaborative robot programming in smart factories," *KI-Künstliche Intelligenz*, vol. 33, no. 2, pp. 151–161, 2019.

[17] J. Horst, E. Messina, J. Marvel, *et al.*, "Best practices for the integration of collaborative robots into workcells within small and medium-sized manufacturing operations," *National Institute of Standards and Technology Advanced Manufacturing Series 100-41*, 21 pages, 2021.

[18] Å. Fast-Berglund, F. Palmkvist, P. Nyqvist, S. Ekered, and M. Åkerman, "Evaluating cobots for final assembly," *Procedia CIRP*, vol. 44, pp. 175–180, 2016.

[19] L. G. Christiernin, "How to describe interaction with a collaborative robot," in *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 2017, pp. 93–94.

[20] F. Zhao, C. Henrichs, and B. Mutlu, "Task interdependence in human-robot teaming," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, IEEE, 2020, pp. 1143–1149.

[21] T. Kopp, M. Baumgartner, and S. Kinkel, "Success factors for introducing industrial human-robot interaction in practice: An empirically driven framework," *The International Journal of Advanced Manufacturing Technology*, vol. 112, no. 3, pp. 685–704, 2021.

[22] E. Ras, F. Wild, C. Stahl, and A. Baudet, "Bridging the skills gap of workers in industry 4.0 by human performance augmentation tools: Challenges and roadmap," in *Proceedings of the 10th International Conference on PErvasive Technologies Related to Assistive Environments*, 2017, pp. 428–432.

[23] C. Giffi, P. Wellener, B. Dollar, H. A. Manolian, L. Monck, and C. Moutray, "Deloitte and the manufacturing institute skills gap and future of work study," *Deloitte Insights*, 2018.

[24] N. Shmatko and G. Volkova, "Bridging the skill gap in robotics: Global and national environment," *SAGE Open*, vol. 10, no. 3, p. 2 158 244 020 958 736, 2020.

[25] D. Chrisinger, "The solution lies in education: Artificial intelligence & the skills gap," *On the Horizon*, 2019.

[26] M. Andrew, T. Marler, J. Lastunen, H. Acheson-Field, and S. W. Popper, *An Analysis of Education and Training Programs in Advanced Manufacturing Using Robotics*. RAND, 2020.

[27] S. El Zaatari, M. Marei, W. Li, and Z. Usman, "Cobot programming for collaborative industrial tasks: An overview," *Robotics and Autonomous Systems*, vol. 116, pp. 162–180, 2019.

[28] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 5537–5544.

[29] J. Huang and M. Cakmak, "Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts," in *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, IEEE, 2017, pp. 453–462.

[30] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blockly goes to work: Block-based programming for industrial robots," in *2017 IEEE Blocks and Beyond Workshop (B&B)*, IEEE, 2017, pp. 29–36.

[31] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd, and D. Franklin, "Evaluating coblox: A comparative study of robotics programming environments for adult novices," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[32] F. Steinmetz, A. Wollschläger, and R. Weitschat, "Razer—a hri for visual task-level programming and intuitive skill parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1362–1369, 2018.

[33] F. Steinmetz, V. Nitsch, and F. Stulp, "Intuitive task-level programming by demonstration through semantic skill recognition," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3742–3749, 2019.

[34] A. Schoen, C. Henrichs, M. Strohkirch, and B. Mutlu, "Authr: A task authoring environment for human-robot teams," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, 2020, pp. 1194–1208.

[35] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production," in *2016 ieee/rsj international conference on intelligent robots and systems (iros)*, IEEE, 2016, pp. 2293–2300.

[36] Y. Gao and C.-M. Huang, "Pati: A projection-based augmented tabletop interface for robot programming," in *Proceedings of the 24th international conference on intelligent user interfaces*, 2019, pp. 345–355.

[37] E. Senft, M. Hagenow, K. Welsh, R. Radwin, M. Zinn, M. Gleicher, and B. Mutlu, "Task-level authoring for remote robot teleoperation," *Frontiers in Robotics & AI*, 2021.

[38] E. Senft, M. Hagenow, R. Radwin, M. Zinn, M. Gleicher, and B. Mutlu, "Situated live programming for human-robot collaboration," in *ACM Symposium on User Interface and Software Technology*, 2021.

[39] V. Dagdilelis, M. Sartatzemi, and K. Kagani, "Teaching (with) robots in secondary schools: Some new and not-so-new pedagogical problems," in *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05)*, IEEE, 2005, pp. 757–761.

[40] S. Ziaeefard, M. H. Miller, M. Rastgaar, and N. Mahmoudian, "Co-robotics hands-on activities: A gateway to engineering design and stem learning," *Robotics and Autonomous Systems*, vol. 97, pp. 40–50, 2017.

[41] D. Mavrikios, N. Papakostas, D. Mourtzis, and G. Chryssolouris, "On industrial learning and training for the factories of the future: A conceptual, cognitive and technology framework," *Journal of Intelligent Manufacturing*, vol. 24, no. 3, pp. 473–485, 2013.

[42] G. Chryssolouris, D. Mavrikios, and L. Rentzos, "The teaching factory: A manufacturing education paradigm," *Procedia Cirp*, vol. 57, pp. 44–48, 2016.

[43] E. Matsas and G.-C. Vosniakos, "Design of a virtual reality training system for human–robot collaboration in manufacturing tasks," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 11, no. 2, pp. 139–153, 2017.

[44] U. Robots, *Universal robots academy*, 2021. [Online]. Available: https://academy.universal-robots.com.

[45] M. Słowikowski, Z. Pilat, M. Smater, and J. Zieliński, "Collaborative learning environment in vocational education," in *AIP Conference Proceedings*, AIP Publishing LLC, vol. 2029, 2018, p. 020070.

[46] P. Fantini, M. Pinzone, F. Sella, and M. Taisch, "Collaborative robots and new product introduction: Capturing and transferring human expert knowledge to the operators," in *International Conference on Applied Human Factors and Ergonomics*, Springer, 2017, pp. 259–268.

[47] M. T. Chi, P. J. Feltovich, and R. Glaser, "Categorization and representation of physics problems by experts and novices," *Cognitive science*, vol. 5, no. 2, pp. 121–152, 1981.

[48] N. R. Council *et al.*, *How people learn: Brain, mind, experience, and school: Expanded edition*. National Academies Press, 2000.

[49] A. A. diSessa, "Knowledge in pieces.," in *Constructivism in the computer age.* Ser. The Jean Piaget symposium series. Hillsdale, NJ, US: Lawrence Erlbaum Associates, Inc, 1988, pp. 49–70, ISBN: 0-8058-0101-4 (Hardcover).

[50] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2016–2021.

[51] N. Fraser, "Ten things we've learned from blockly," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, IEEE, 2015, pp. 49–50.

[52] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[53] A. J. Ko and B. A. Myers, "Human factors affecting dependability in end-user programming," in *Proceedings of the first workshop on end-user software engineering*, 2005, pp. 1–4.