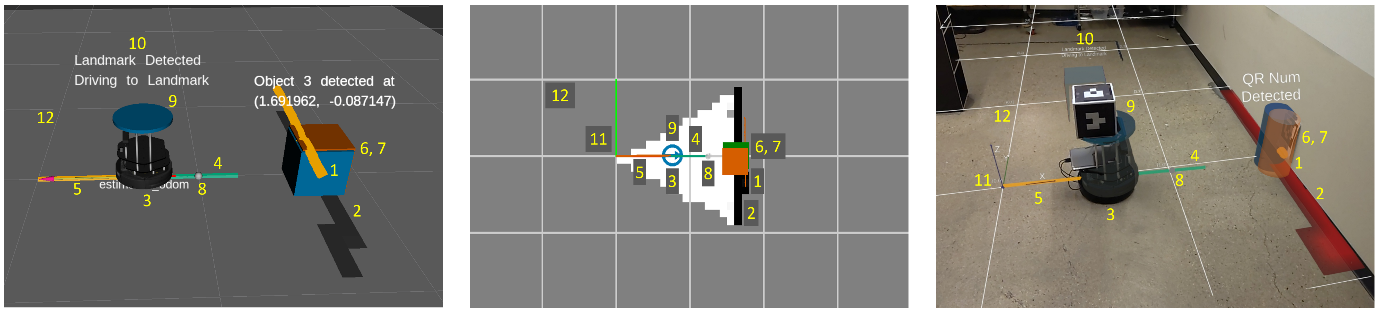
Advancing the Design of Visual Debugging Tools for Roboticists

Bryce Ikeda
Department of Computer Science
University of Colorado Boulder
Boulder, Colorado
bryce.ikeda@colorado.edu

Daniel Szafir
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina
daniel.szafir@cs.unc.edu



(a) RViz (b) 2D GUI (c) AR

Fig. 1: In this paper, we explore how three different visual tools may aid robotics debugging. Shown above are identical snapshots in time during a robot object detection task. Each tool varies in their display of the robot's sensor and state information, annotated with yellow numbers that are described in Table I.

***Abstract*—Programming robots is a challenging task exacerbated by software bugs, faulty hardware, and environmental factors. When coding issues arise, traditional debugging techniques such as output logs or print statements that may help in typical computer applications are not always useful for roboticists. As a result, roboticists often leverage visualizations that depict various aspects of robot, sensor, and environment states. In this paper, we explore various design approaches towards such visualizations for robotics debugging support, including 3D visualizations presented on 2D displays, as in the popular RViz tool within the ROS ecosystem, visualizations in a two-dimensional graphical user interfaces (2D GUI), and emerging immersive three-dimensional (3D) augmented reality (AR). We present a qualitative evaluation of feedback gathered from 24 roboticists across two universities who used one of these debugging tools and synthesize design guidelines for advancing robotics debugging interfaces.**

***Index Terms*—Augmented Reality (AR); Mixed Reality (MR); visualization; interface design; robots; debugging; HRI; HCI**

I. Introduction

Programming robots is difficult. In addition to standard challenges faced by any computer programmer, such as syntax, logic, compilation, or runtime errors, roboticists must also deal with complications caused by system variability from environmental factors and unreliable hardware. Typical debugging techniques, such as printing raw data to the console or log files, can be tedious and often confusing as robots may contain a variety of motors and sensors with a range of complex data types. To address this issue, roboticists often make use of visualizations of robot state data. For example, end effector transformations can be difficult to interpret and validate via logs of matrix data, but when inspected through 3D visualizations, can be easily confirmed.

While there has been significant research in the human-robot interaction (HRI) community that explores visualizations of robot state and for end-users [1]–[7], to date there has been limited work specifically focusing on visualization design for roboticists. This paper seeks to address this gap by treating roboticists themselves as the "human" component of human-robot interaction and investigating how different modalities and presentations of robot data may support robot debugging.

One of the most widely used tools for robotics programming across both academic research and industry settings is the Robot Operating System (ROS). RViz, a central component of the ROS ecosystem, enables roboticists to visualize 3D data on a 2D monitor [8]. Examples of these visualizations include point clouds, object affordances, and robot models [9]–[11]. While commonly used, little work has been done to evaluate the design choices of this tool. For example, RViz contains a clear design flaw in its use of rainbow color schemes for data encoding [12], [13]. Aside from RViz, other roboticists leverage 2D visualizations. For example, Zaman et al. [14] and Aini et al. [15] observed their mobile robots from a top down view to visualize environment mapping, location, and sensor data. While 2D GUIs are prevalent in literature, we have found limited work focusing on their design for development and debugging or enabling reasoning over potential design trade-offs between RViz and 2D GUI approaches.

In addition to RViz and 2D GUIs, 3D immersive AR visualizations are emerging as a third modality for visualizing robot data for end-users [1], [2], [5]–[7], [16], [17]. To date, there have been no detailed explorations investigating the use of AR visualizations for robotics debugging. However, prior work by Collett and MacDonald found benefits to using 2D AR annotations in their ARDev system for robot debugging [18]. While promising, ARDev could only provide 2D visualizations displayed on a TV screen (i.e., 2D AR overlays [19]), which lack stereo depth cues, has a fixed field-of-view, and require that users translate their perspective into a different context (a top-down view). Such setups separate data from its context and can reduce a user's performance on tasks related to spatial understanding and robot control [7], [20], [21]. Modern AR head-mounted displays (ARHMDs) can eliminate these issues by projecting visualizations directly within a single, unified context in the user's real environment. Thus we believe ARHMDs hold great potential for robotics debugging.

Contributions: In this work, we conceptually replicated [18] to investigate robotics debugging. We performed a qualitative evaluation, inspired by the design study methodology of [22], to compare debugging between RViz, a 2D GUI, and AR with 24 roboticists. Our novel contributions include: (1) exploring AR with a modern ARHMD (vs 2D AR annotations on a top-down screen as in [18]), (2) investigating debugging in the context of modern robotics programming with ROS ([18] did not use ROS), (3) comparing AR to an industry-standard tool RViz (vs [18] which only compared fixed screen AR to a 2D GUI), and (4) collecting data from substantially more roboticists (24 in our study vs 5 in [18]) with more experience (29% of our users had $\geq$ 3 years vs in [18] all had $\leq$ 3 years). Finally, we present a thematic analysis of our findings and offer design guidelines for developing future systems aimed at advancing robotics by supporting roboticists themselves.

II. Related Work

In this section, we discuss prior work related to debugging, human-robot interfaces, and augmented reality.

A. Debugging Code

Debugging in its most basic terms is defined as "the attempt to pinpoint and fix the source of an error" [23]. To better understand this process, Lawrence et al. applied the *information foraging theory*, that models the programmer as a predator following information scents while navigating through their code topology to find their prey: the bug in their code [24]. In practice, debugging is a complex process influenced by a user's experience level and the tools they have at their disposal. Various techniques have been developed to aid with this process, such as program slicing, answering reachability questions, and answering why lines [25]–[27]. However, these approaches are difficult to implement on distributed systems, such as robots, where two executions of the same program may provide different results due to communication delays, hardware errors, or environmental instability. Other researchers have also used time-coded example data and a timeline interface to program and debug code visually [28]. Still, these interfaces require that the user gather large amounts of high quality video data, and inhibits the generalization of their program to varying tasks. Our research explores how visual debugging tools may fill these gaps by providing real-time debugging information to help roboticists quickly pinpoint and fix errors.

B. Human-Robot Interfaces & Augmented Reality

To date, most HRI interface research has focused on supporting end-users with non-programming tasks such as enhancing situational awareness or system usability (see [29] for a survey and relevant links to data visualization). Within this space, researchers are increasingly leveraging AR as a tool to improve user interactions with robots ([30] & [31] provide surveys). Prior to the introduction of modern ARHMDs, researchers developed applications that overlaid data on 2D screens to better communicate the state of mobile robots, manipulators, and robotic swarms [3], [18], [32], [33]. Such 2D annotations have shown promise, but are limited by a lack of stereopsis as a depth cue and fixed display windows. It also forces users to shift perspectives between their real-world view and a screen. More recent work has investigated the use of ARHMDs to reduce or eliminate these issues for end-users, finding that immersive AR can produce benefits for robot control [5], [17], [34], [35] and robot learning [16], [36]. While this research exhibits how AR may aid end-users with understanding system behaviors, they do not focus on users who are fixing their robot code.

Other work has explored AR systems with the intention of robotics education. For example, researchers have studied how AR tablets may assist K–12 students in understanding robots [37]–[39]. While this helps provide insight into how to educate new roboticists, our focus is roboticists that are already experienced developers. Similar to the AR debugging system we develop in this work, Muhammad et al. and Cleaver et al. created an AR framework for visualizing sensor data obtained from a robot [39], [40]. While robot debugging is identified as a potential application for their system, their work focuses primarily on communicating robot motion intent and improving robotics education [39], [40]. The only prior research we can identify that is directly aligned with our goals is Collett and MacDonald's work on ARDev [18], as described earlier. We extend [18] by investigating modern immersive AR, using current robotics programming (i.e., ROS) contexts, and including comparisons to industry-standard tools (i.e., RViz), while drawing on interface design principles from Human-Computer Interaction (HCI) [41], [42] and principles from Visualization (VIS). One such principle includes sensemaking—the process of how humans work with information [43]—to explore how we may further improve debugging tools for roboticists and advance progress in the field overall.

III. Comparing Visual Robotics Debugging Tools

In this work, we compared existing and emerging visualization tools that support robotics debugging. We ran a qualitative expert evaluation with 24 roboticists across two universities. Below, we provide study and implementation details.

A. Study Design

Our study was a 3×1, between-participants design where each participant used one of three visualization aids (RViz, 2D GUI, or AR) when programming a robot to complete two tasks. Our overall approach was akin to that of a Design Study [22] to gain expert feedback on each visual debugging tool.

B. Programming Tasks

Our study presented roboticists with two programming problems, both of which were modified versions of tasks initially proposed by Collett and MacDonald as representative of standard applications robot developers typically code: a *detection* task and a *finder* task [18]. Participants completed these tasks in the context of programming and debugging a TurtleBot 2, a low-cost, differential drive mobile robot commonly used as an entry machine by roboticists.

For the detection task, participants were given 30 minutes to program a TurtleBot to detect a QR code 1.5*m* in front of the robot's camera, drive up to it, and stop within 0.6*m* of the QR code. For the finder task, participants were given 1 hour to program a TurtleBot to search a room for a QR code placed in an unknown location, drive up to it, and stop within 0.6*m* of the QR code. These tasks required object detection and navigation thus requiring the programmer to take advantage of multiple forms of data and control sequences. We provided participants with a pre-configured ROS directory containing an autonomous robot framework that included files for EKF-SLAM for localization, A* for path planning, Pure Pursuit for path following, and OpenCV to track QR codes. Participants coded within this directory using either Python or C++.

C. Robot Debugging Data Visualization Tools

As participants completed their programming tasks, they inevitably made errors that needed to be debugged. Participants were provided one of three different visual debugging tools that highlight various approaches for how interfaces might display robot data. This enables a comparison of a breadth of design factors such as 2D (2D GUI) vs 3D (RViz) vs immersive 3D (AR) and separate visualization context (RViz and 2D GUI) vs integrated context (AR).

- **RViz** is a commonly-used 3D visualization tool for robotics. It can display a robot model, environment maps, and sensor data via a graphical user interface on a monitor (Figure 1a). To interface with RViz, participants used their mouse left, right, and middle clicker to move the virtual camera and edit the visualizations in the environment.
- We created a **2D GUI** as a replication of Collett and MacDonald's 2D GUI from Player/Stage [18]. It displays similar visualizations as RViz and the immersive AR tools, but from a top-down perspective (Figure 1b). To interact with this tool, the *w*, *a*, *s* and *d* keys enabled traversal in the vertical and horizontal directions while the keys *i* and *o* enabled zooming in/out.
- We developed an **immersive AR** debugging interface that leverages the HoloLens ARHMD to project *in situ* visualizations within the wearer's workspace (Figure 1c). The HoloLens was untethered from the development computer, allowing users to freely move through the environment and see visualizations from any angle. It also displayed coordinates and measured distances when the user interacted with the map grid through the HoloLen's air-tap feature. This was an added feature recommended by Collett and MacDonald's work and was not implemented as a feature on the RViz or the 2D GUI because roboticists do not typically use these tools on these applications. Our AR design also addresses technological limitations of Collett and MacDonald's work (i.e., using 2D AR overlays on separate screens vs an ARHMD). We used ROS# [44] to send the data from ROS to our AR application.

All three interfaces provide a different medium for visually representing data (2D, 3D on a 2D screen, and immersive 3D). To reason over how robot data might be effectively translated to its visual representation at the interface level, Collett and MacDonald identified four principle data types: *geometric* data with an intrinsic visual representation using points, lines and shapes (e.g., a robot model to represent position); *vector* data with a magnitude and an orientation (e.g., a planned path between two points); *scalar* data with only a magnitude (e.g., a robot's speed or mass); and *abstract* data without an intrinsic visual representation (e.g., a robot's state) [18]. We extend this framework by splitting *abstract* into four categories: *text* data expressed verbally (e.g., grid coordinates); *raster* data with a grid of scalar values that can include multiple information bands (e.g., an occupancy grid); *categorical* data with distinct attributes that belong to a specific concept (e.g., a robot's state); and *boolean* data that is true/false, yes/no, or 0/1 (e.g., the result of a conditional). These primitive abstractions (i.e., "data types") can inform a designer's choice of visual encodings when referencing established visualization principles. For example, categorical information can be readily encoded using color, specifically by multiple hues [45]. As robot data may be visualized in a multitude of ways, determining the high level data types may aid developers with identifying corresponding visualization techniques that most clearly communicate important information to the users. See Table I for a complete list of rendered data types in each visual debugging tool.

D. Environment

Two locations were used, one robotics laboratory at the University of Colorado at Boulder and the other at Colorado School of Mines. These locations were chosen to simulate environments participants typically code and debug robots in. Both rooms measured approximately $5m \times 5m \times 3m$, each containing one workstation for the programmers to code on. A $0.3m \times 0.3m \times 0.9m$ box with a $0.18m \times 0.18m$ QR code

TABLE I: The robot sensor and state data provided to the participant, its associated ROS message type, data type, visualization description and the debugging tool it was displayed in. The legend corresponds to its respective value in Figure 1.

Data	Message Type	Data Type	Description of Visualization	RViz	2D GUI	AR	Fig. 1 Legend
Laser Scan	sensor_msgs/LaserScan	Vector	LIDAR data from the ASUS XTION	✓	✓	✓	1
Occupancy Grid	nav_msgs/OccupancyGrid	Raster	Obstacles in the environment via the Occupancy Grid mapping algorithm	✓	✓	✓	2
Position	nav_msgs/Odometry	Geometric	The position and orientation of the Turtlebot	✓	✓	✓	3
Waypoint Path	nav_msgs/Path	Vector	The planned path using A* on the user's input waypoints	✓	✓	✓	4
Waypoint Path History	nav_msgs/Path	Vector	The previously traversed path of the robot	✓	✓	✓	5
Raw QR Position	perception/Landmarks	Geometric	The detected position of a QR code using OpenCV	✓	✓	✓	6
Estimated QR Position	perception/Landmarks	Geometric	The estimated position of the QR codes via EKF SLAM	✓	✓	✓	7
Look Ahead Point	geometry_msgs/Pose	Geometric	The point the Pure Pursuit Tracking algorithm is planning towards	✓	✓	✓	8
Robot State	visualization_msgs/Marker	Categorical	The user defined color of the robot to portray robot state	✓	✓	✓	9
Camera Feed	sensor_msgs/Image	Raster	The camera feed from the robot displayed on the computer screen	✓	✓	✓	–
Text (Superimposed)	visualization_msgs/Marker	Text	Debugging text data output in the users debugging tool environment	✓	×	✓	10
Text (Terminal)	–	Text	Debugging text data output in the Terminal using rospy.loginfo and ROS_INFO	✓	✓	✓	–
Origin Axis	–	Vector	The x, y and z direction of the coordinate plane (see limitations section)	×	✓	✓	11
Map Grid	–	Vector	A grid of 1m x 1m boxes (only the HoloLens has coordinate labels)	✓	✓	✓	12
Distance Tool	–	Text	Users obtain the distance between two selected points on the Map Grid	×	×	✓	–
Coordinate Tool	–	Text	Users obtain the coordinates of points selected on the Map Grid	×	×	✓	–
Landmark Detected	–	Boolean	True when a QR code is being detected, False otherwise	–	–	–	–
Waypoints Sent	–	Boolean	A helper variable set by the participant in their code	–	–	–	–

attached to it was used as a target object for the programming tasks (described in more detail below). Both rooms were controlled during the experiment (i.e., only the participant was in the room with no other distractions or people).

E. Participants & Procedure

We recruited a total of 27 participants (22 male, 4 female, and 1 prefer not to say), 16 from the University of Colorado Boulder and 11 from the Colorado School of Mines. This study was approved by our university IRB. Three of the participants' data could not be used due to errors that occurred with their assigned debugging tool during their session, leading to our total analyzed dataset having 24 participants. The average age of the participants was 24.5 years old (SD = 3.50), with a range of 18–35. To qualify for our study, participants were required to have experience creating at least one robotics project containing one or more ROS package. Of the 24 users, seven (29.17%) reported working with ROS for three or more years. Nineteen (79.17%) of the participants reported having worked with a mobile robot in the past. On a five-point scale, users reported their average familiarity with RViz as 3.29 (SD = 1.32), with a 2D GUI as 1.63 (SD = .86), and with AR as 2.00 (SD = 1.20). Their average experience using ROS was 1.96 years (SD = 1.08). We believe our sample of participants illustrate a broad distribution of roboticists with a wide range of experience levels compared to prior work.

Each participant's session consisted of five phases: (1) introduction, (2) training, (3) task one (detection), (4) task two (finder), (5) conclusion. (1) First, participants were given a consent form to read and sign, and then were brought into the task space. (2) Next, they were given their assigned debugging tool. If they were selected for the AR interface, they were quickly navigated through the HoloLens menu to calibrate their eyes for the device. They were then given 5 minutes to read identical instruction sheets outlining the two tasks, the message types, variable names, functions, and launch files provided. Then, the participants were given a tutorial on how to navigate and use their assigned debugging tool, followed by a 15*s* video depicting the predefined visualizations participants had access to in their debugging tool. They were also shown a 10*s* example video showing the completion of task one, but were told they could accomplish the task however they saw fit. Finally, they were given time to ask clarifying questions. (3) Participants began the detection task, and finished once the robot reached within 0.6*m* of the QR code or when their 30 minute time limit elapsed. If the participants were unable to finish task one but were close to completion, the proctor guided the user's code to a working product. This was done so the users could work with code they wrote and were familiar with going into the finder task. If the proctor deemed their code too far off from a working solution, they were given a working file. (4) Participants were then shown a 20*s* video of a visual representation of how the finder task could be accomplished. However, participants were told they could solve it however they saw fit. Participants then began the finder task and finished once the robot reached within 0.6*m* of the QR code or when 1 hour elapsed. (5) After the second task, a survey was administered to understand the participants experiences (see Appendix A). Following the survey, we collected open-ended verbal feedback regarding their assigned debugging tool. Finally, participants were debriefed and then compensated with a $40 gift card for their time.

F. Analysis Method

We analyzed the roboticists' responses using thematic analysis, a qualitative research method used for identifying, organizing, and reporting themes and patterns within a data set [46]. This type of analysis is useful for examining the

TABLE II: A summary of our themes and design guidelines to inform future visual debugging tools.

Theme	Sub-Themes	Design Guidelines
Sensemaking via Intuitive Visualizations	Understanding Context Understanding Robot Cognition Hypothesis Development Knowledge of Past and Future States Color Objects and Labels	A programmer's sensemaking process can be aided by intuitive visualizations that properly encode data relevant to the current task. For example, visualizations such as waypoints or spatially recognizable object locations were intuitive for a user's understanding of navigational cues. This provided users with a clearer connection between their code and the behavior of the robot, thereby improving their debugging process.
Visual Clutter	Mitigate Occluding Visualizations Reducing Print Statements	Minimizing visual clutter helps users identify and synthesize important information. An example of this would be to develop context-aware debugging tools that group visualizations by the task they inform, rather than all at once.
Cognitive Load	Visual Learning Superimposed Visualizations Perspective Taking Tool Learning Curves	Debugging interfaces should be designed to cohesively integrate information from disparate data streams to reduce a user's cognitive load. In our study, AR users were frustrated by switching between the real environment and the computer terminal. Improvements might be to add a virtual terminal in situ within the AR environment or to find ways of embedding errors and warnings within AR.
User Interfaces	Simplicity vs Customization Interactive User Interfaces 2D versus 3D Interfaces	More research is needed to understand how debugging aids may strike a balance between simplicity and useful features.

perspectives of participants and summarizing key features of a large data set. As described by Braun and Clarke, thematic analysis is done in six phases: (1) familiarizing yourself with the data, (2) generating initial codes, (3) searching for themes, (4) reviewing potential themes, (5) defining and naming themes, (6) producing the report [46]. From our analysis, we define four key design themes with 19 sub-themes (see §IV).

IV. Design Themes

Of the 24 participants, six (75%) using RViz, four (50%) using the 2D GUI and five (62.5%) using AR were able to complete task one. For task two, five (62.5%) of the participants in each condition successfully completed the task. At a high level, visual debugging tools can benefit a roboticist's understanding of their code, while violations of HCI heuristics and VIS principles may reduce the effectiveness of such tools. The four major design themes synthesized from our qualitative data include Sensemaking via Intuitive Visualizations, Visual Clutter, Cognitive Load, and UI (Table II). Below, we discuss only the first three themes as the last theme involves lower-level aspects of specific interface implementation.

A. Sensemaking via Intuitive Visualizations

The principle of sensemaking is defined as "the process of searching for a representation and encoding data in that representation to answer task-specific questions" [47]. Visual debugging tools can ameliorate sensemaking during debugging when there is a clear connection between the tool's visualizations, its physical environment and a programmer's code. Our analysis revealed six major aspects of how visualizations affected sensemaking during debugging: contextual understanding, building models of robot cognition, hypothesis development regarding errors, knowledge of past and future states, color encodings, and object and label encodings.

1) Understanding Context: An important aspect of programming a robot is understanding its sensor reach and operational space. For example, research shows that visualizations that provide the location where sensors are collecting data help operators understand the context of their task, thereby improving performance [7], [21]. Similarly, in our study, roboticists perceived that visual debugging tools helped them understand how the internal workings of the robot related to the real world:

P26[RViz]: *"The visualizations of the paths and targets helped me connect the software tools to the real world when the real robot might have a different perception of the space than I believe."*

Other research has looked into providing visualizations of spatial regions to communicate sensed, inherent and user-defined spatial regions to users [4], [48], [49]. In our study, four (50%) of the RViz users and five (62.5%) of the AR users expressed a greater comprehension of the robot's spatial environment when this data was displayed. For instance, the location of waypoints and the size of the environment helped participants develop and debug their code.

P6[RViz]: *"I needed the size of the terrain to generate waypoints to explore. RViz gave me a sense of how large the environment was."*
P24[AR]: *"I believe [AR] can help programming robots, where spatial information is very important to contextualize what you are doing with your actual code instead of sitting there looking at numbers on a console...it can be easier to see what the robot is thinking rather than what you think the robot is thinking..."*

Although one of the 2D GUI users expressed a liking towards their tool, others had more trouble understanding their environment. When asked what frustrated them the most about their debugging tool, they responded with:

P3[2D GUI]: *"Mostly my lack of understanding of the relationship between the waypoints I was sending and the robot's behavior - I started out assuming the robot would respond to the waypoint publishing in a particular way, and it took me a while to figure out that I had misunderstood. The GUI didn't help in that regard."*
P5[2D GUI]: *"It would be nice to always know what the bounds of the robot are."*

2) Understanding Robot Cognition: One benefit to visual debugging tools is the user's increased ability to understand and confirm a robot's internal "thoughts"—how the robot is seeing and thinking about the world—which requires synthesizing information about robot hardware and software states. For instance, the following quote is representative of various incidents where visualizations helped users identify mismatches between robot cognition and the real world:

P12[AR]: *"[AR] helped me find out that the robot odometry was not accurate and the robot needed to be restarted."*

Eleven (45.83%) of the total users noted this benefit particularly in debugging activities related to robot perception:

P16[RViz]: When asked what they liked about RViz: *"Visualizing and seeing what the robot can see in order to help us make better decisions."*
P5[2D GUI]: *"[The 2D GUI] helped me see if the robot was recognizing objects/qr markers in the environment."*
P25[2D GUI]: *"It was great to see that the vision and SLAM system was actually working."*
P10[AR]: *"I was initially not sure what was happening because I didn't write the planner, so when I saw that okay it is detecting [the QR code] and trying, going towards it, it helped me understand what it's thinking."*

This feedback is particularly insightful as to the effect visualizations may have on robot development teams. While one part of the team may work on a robot's autonomous stack, others may work on the user interface, both having limited knowledge of the others work. By providing robot program validation through informative visualizations, we may be able to promote cohesion and understanding across developers.

3) Hypothesis Development Regarding Errors: An important part of debugging is the ability to develop a hypothesis explaining why code is not working and where the bug may be. Ten (41.67%) of the participants noted their debugging tool helped them gain insights into their code:

P26[RViz]: *"I could rule out steps between the real robot and the program—the robot might not be facing the target, and so the debugging tool would show the target in the space it perceives."*
P13[2D GUI]: *"It was nice to see where the robot was going with the code I had written so I could know better how to change my code."*
P19[AR]: *"[AR] helped me visualize/create a mental map and helped me see the gap between the code and where the robot went in reality."*

4) Knowledge of Past and Future States: An important part of the debugging process is the user's ability to predict unwanted behavior, thereby understanding what to prevent from happening. Predictions of future behavior can be further informed by replaying a robot's behavior from the past. During the development of task two, eight (33%) of the participants specifically asked for a feature that would allow them to visualize the robot traversing all of their programmed waypoints before execution, as well as a feature to review the robot behavior from the past.

P25[2D GUI]: *"It would have been nice to have a standard video player where you could drag it, drag time, stop, play, reverse. Being able to play through that rather than rosbag play..."*
P27[2D GUI]: *"I just wanted to show my waypoints on the GUI tool before executing the plan."*
P9[AR]: *"I'd use the holographic Turtlebot to display programmed behavior before executing the behavior on the real robot. Adding an option to speed up trajectory playback could also be helpful while debugging (2x, 4x etc)."*

While replaying data is a feature commonly performed by a tool called rosbag, **P25** sees a benefit to providing this feature through a GUI. Developers can do this using a package in ROS call rqt_tools. However, only three (12.5%) of the participants reported using the plugins provided by rqt_tools for debugging. As a result, it is worth investigating how to properly integrate this feature into commonly used debugging tools.

5) Color: The use of color to encode data can clarify or inhibit a user's understanding of their task. For example, rainbow color maps hinder a user's ability to perform perceptual ordering of colors and to see important details in the map [12], [29]. Certain color choices may also create accessibility issues for color blind users (to put this issue into perspective, nearly 1 in 12 males are colorblind [50]). While the effect of color maps were not specifically investigated in our study, in each of the debugging tools, the waypoint path, waypoint path history, laser scan, robot state and QR positions were all displayed using a color palette accessible to color blind users, a design feature we believe could improve the accessibility of existing tools such as RViz. However, we one user wanted to keep color consistent with what they were used to seeing (e.g. red for errors, yellow for warnings) and another was confused by the colors in general.

P17[2D GUI]: *"That's how ROS error messages are right? Errors are red in color, ROS warnings are yellow in color."*
P21[2D GUI]: *"I wasn't 100% sure what the different colors were but I could make a good assumption."*

Based on this feedback, it may not be enough to provide color blind friendly visualizations unless all users can intuitively understand the information the colors are encoding. For example, keeping in line with the standards understood by **P17**, but also adhering to color blind accessible color palettes, one might use a reddish purple to signify paths a robot cannot reach, yellow to represent paths in progress and bluish green to display successful paths traversed [50]. Furthermore, while labeling colors may explain their meaning, a more intuitive approach could be to incorporate geometric shapes such as arrows to display direction or animations (e.g., blinking) to portray uncertainty [51], [52].

6) Objects and Labels: When visualizations were not labeled, such as the axis orientation or the grid size, five (62.5%) 2D GUI users felt they were missing important information. In the RViz condition, due to a bug where the robot transform axis was displayed rather than the origin axis, two (25%) participants reported having difficulty understanding the axis orientation initially.

P20[RViz]: *"I wish there was an X/Y axis so I could get my bearings* *better."*

P11[2D GUI]: *"I have a feeling it's a part of the experiment but a more descriptive setup would have been nice to have (axis labels, grid step size, maybe even a real time pose readout would have been nice)."*

P17[2D GUI]: *"Understanding the x and y axis of the robot. If this was provided in a legend, it would have been easier to code rather than to go with hit and trial to see what is what."*

In these cases, participants had difficulty understanding the direction of the x and y axis. Similarly, while each grid square was $1m \times 1m$, participants had difficulty inferring distances just by looking at the grid map. In the AR tool, the axis of orientation and grid were clearly labeled with axis directions and grid step sizes. This clarified the scene for users with one of the participants stating:

P4[AR]: *"It gave me the coordinate system right away so I didn't have to infer that from robot's behavior."*

Research has found evidence supporting that icons need to be supported by text to promote user understanding of its functionality [53]. Similarly, a clearer representation of spatial indicators can be developed through labels that clearly convey what they mean to the user.

B. Visual Clutter

While visualizations can convey meaningful information to users, too many visual artifacts can quickly inhibit a user's understanding of what is being displayed. This issue is commonly referred to as visual clutter or overdraw and can decrease information recognition, scene segmentation, and visual search performance [54]. Our results offer two suggestions:

1) Mitigate Occluding Visualizations: One RViz user had an issue of previous paths blocking their view of future ones.

P26[RViz]: *"The previous paths are still visible, impeding my ability to see the current path."*

Similar issues with occluding or cluttered visualizations occurred when participants coded waypoints with step sizes smaller than the robot. Since waypoint locations were displayed on the floor of the scene, future waypoints were sometimes rendered beneath the robot's digital twin. This prevented participants from visualizing the list of waypoints being traversed by the robot. In one case, this prevented a participant from realizing waypoint visualizations were a feature being displayed.

P14[RViz]: *"I don't know if it is possible, but I think it would be nice to show the next waypoint the Turtlebot is going to move to on RViz."*

For 2D interfaces, visual clutter is difficult to prevent. However, solutions from computer graphics could be applied where the salience of data can be manipulated to direct visual attention [55], [56]. For example, the path to the next waypoint can be rendered opaque while the path history can be displayed using a greater level of transparency as time goes by [29]. Transparent digital twins of the robot can also be used so that it does not obstruct the view of waypoint visualizations. Since 3D visualization tools have the ability to re-position visualizations by traversing the z-axis, future and past paths can also be displayed at a height that won't be covered by the robot.

2) Reducing Print Statements: Another form of visual clutter in programming can present itself as overload of output print statements in a terminal. Twenty (83.3%) of the participants reported using print statements as a form of debugging their code. While printing variables is an encouraged form of debugging that can be both quick and informative, multiple participants valued visualizing the data as opposed to printing it.

P18[RViz]: *"It helped me not need to print as much to the terminal because I could see what the robot was thinking/doing."*

P20[RViz]: *"I needed less print statements which meant that the terminal filled up slower."*

P5[2D GUI]: *"[The visualizations] allowed me to rule out particular possibilities as to why the robot was not moving. I believe that this was faster than it would have been if I had to print out particular data points to locate the reason for my error."*

In particular, while **P20** still used print statements, the participant benefited from seeing the printed output of only the most relevant information in a less cluttered terminal.

C. Cognitive Load

Cognitive load is defined as a "multidimensional construct representing the load that performing a particular task imposes on the learner's cognitive system" [57]. Naturally, the process of debugging can involve a high cognitive load varying based on the user's experience level [58]. We found three primary ways visualizations may reduce the mental load and effort that it takes to understand and fix issues in code.

1) Visual Learning: We found that visualization tools may be particularly useful for programmers who identify themselves as visual learners. By seeing data in the context it is meant to portray, eight (33.3%) of the participants commented they had an easier time understanding the program without looking directly at the code.

P26[RViz]: *"I am a visual learner, and so the visualizations of path and targets helped me to connect the software tools to the real world."*

P11[2D GUI]: *"I'm a visual learner so this tool definitely made my thought process easier to work through."*

P10[AR]: *"It helped me understand if my code was working without looking through the code."*

2) Superimposed Visualizations: A key feature of our AR system is its ability to overlay visualizations *in situ* in the user's real environment. Four (50%) of the AR users specifically noted the superimposed visualizations reduced their perceived mental effort when trying to understand spatial relationships.

P9[AR]: *"The superimposed graphics made understanding robot behavior and its operational space very easy."*
P12[AR]: *"The biggest aspect I liked was when you could see where the grid lined up with the robot which is something you don't get with traditional RViz when visualizing the grid."*

3) Perspective-Taking: Perspective-taking involves a user's ability to understand a robot's behavior within the context of the robot's environment to achieve situational awareness, thereby improving performance on certain tasks [59], [60]. AR has improved perspective-taking during teleoperation tasks [7], and the same may hold true for debugging. Two (25%) of the AR users commented on frustrations with switching between the HoloLens and the computer while debugging their code.

P9[AR]: *"Debugging with the HoloLens wasn't as seamless because I had to keep turning around if I needed to get any information from the scene. E.g.,: I turned around to view the scene after running my code for the first task, waiting for the robot to do something, but I didn't realize my code had thrown a syntax error which was displayed in the terminal."*
P10[AR]: When asked what about the tool frustrated them: *"Switching between the HoloLens and back [to the computer]."*

Future AR systems may address these issues by embedding a programming console as an AR window to remove any reliance on the computer terminal. Other areas of research might look into naturally embedding errors and warnings within AR. From another perspective, one experienced RViz user commented that the most difficult aspects of RViz is moving the camera around to get different views of the scene. This prevents the user from viewing the robot from a visually intuitive position, thereby increasing the chance of errors in perspective-taking.

P6[RViz]: *"One thing I don't like about [RViz], is if you are moving in a large area, you have to reset the camera and that is always a pain. It never seems to reset it correctly. It is unnecessarily difficult."*

V. Takeaways and Opportunities

In this research, we studied how 24 roboticists used one of three different visualization tools to assist during their coding and debugging process. From our analysis, we identified three design guidelines in regards to visual debugging tools. First, a programmer's sensemaking process can be aided by intuitive visualizations that properly encode data. Second, by minimizing visual clutter, users have an easier time identifying and synthesizing important information. Lastly, debugging interfaces should be designed to cohesively integrate information from disparate data streams to reduce cognitive load. One example of how to apply our findings is to develop context-aware debugging tools that group visualizations by the task they inform, rather than providing all the information at once. Such a design would allow the user to rotate between visualization groupings that provide only the information relevant to the category they effect. For instance, engineers debugging self-driving cars may need feedback on the autonomous decision making process, such as why a car has stopped. Thus, an interface might group feedback from the perception system, highlighting important information within immediate proximity of the car such as detected stop lights, incoming cars and pedestrians. This will result in reducing the salience of irrelevant data.

A. Limitations

While our study provided us with several new insights for visual debugging systems, we note that our approach is not without limitations. For example, the tasks in our study, which replicated the robot tasks in [18], principally involved localization and navigation. We believe these are representative of tasks many roboticists encounter at some point in their career, but note there are many other important robot tasks (e.g., manipulation) that future work may examine from a debugging perspective. We also keep in mind that our choice of robot platform may have introduced certain biases regarding the data and visualizations that our participants found helpful (e.g., differences might be found with legged platforms). Additionally, we restricted participants to a 2 hour task time, an artificial limitation compared with general robotics development and debugging. Future work might perform longitudinal studies to examine how tools may support debugging across greater timescales. While we strove to recruit a diverse set of users, we ultimately had a majority male sample, reflective of the current robotics population (e.g., see [61] where only 23 roboticists out of 444 individuals were female). Within our sample, user experience levels with their assigned debugging tool varied. Differences in participant experience potentially affected their performance and subjective feedback, although such an imbalance is unavoidable given that RViz is the default visualization tool for ROS. After conducting our study, we found an error in our RViz condition where we displayed the robot transform axis rather than the scene origin axis shown in the other two conditions. This affected user feedback as described in §IV-A6. Finally, since our thematic analysis was based solely on subjective feedback from participants, future work could obtain quantitative data to further back up the application of our design guidelines.

B. Conclusion

By understanding how design aspects improve user experiences, we have an opportunity to improve the debugging capabilities of expert roboticists, while also reducing the barrier to entry for new ones. While this may seem obvious—after all, the HRI community commonly takes a user-centered design approach when developing systems for end-users—we believe it is critical that we start applying these same ideas when developing our own tools. Our hope is that this initial exploration will provide a starting point for future research into the advancement of effective visual development and debugging tools for roboticists.

VI. Acknowledgment

This work was supported by the NSF under Award # 1764092. We thank Tom Williams for his help with this work.

References

[1] Michael Walker, Hooman Hedayati, Jennifer Lee, and Daniel Szafir. Communicating robot motion intent with augmented reality. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, page 316–324, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Eric Rosen, David Whitney, Elizabeth Phillips, Gary Chien, James Tompkin, George Konidaris, and Stefanie Tellex. *Communicating Robot Arm Motion Intent Through Mixed Reality Head-Mounted Displays*, pages 301–316. 01 2020.

[3] Fabrizio Ghiringhelli, Alessandro Giusti, Jerome Guzzi, Gianni Di Caro, Vincenzo Caglioti, and Luca Maria Gambardella. Interactive augmented reality for understanding and analyzing multi-robot systems. 09 2014.

[4] Tomáš Kot and Petr Novák. Utilization of the oculus rift hmd in mobile robot teleoperation. In *Applied Mechanics and Materials*, volume 555, pages 199–208. Trans Tech Publ, 2014.

[5] Michael E Walker, Hooman Hedayati, and Daniel Szafir. Robot teleoperation with augmented reality virtual surrogates. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 202–210. IEEE, 2019.

[6] Christopher Reardon, Kevin Lee, John G Rogers, and Jonathan Fink. Augmented reality for human-robot teaming in field environments. In *International Conference on Human-Computer Interaction*, pages 79–92. Springer, 2019.

[7] Hooman Hedayati, Michael Walker, and Daniel Szafir. Improving collocated robot teleoperation with augmented reality. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, page 78–86, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60(2):337–345, 2015.

[9] Stephen Hart, Paul Dinh, and Kimberly A Hambuchen. Affordance templates for shared robot control. In *2014 AAAI Fall Symposium Series*, 2014.

[10] Sebastian Pütz, Thomas Wiemann, and Joachim Hertzberg. Tools for visualizing, annotating and storing triangle meshes in ros and rviz. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2019.

[11] D Chikurtev, I Rangelov, N Chivarov, E Markov, and K Yovchev. Control of robotic arm manipulator using ros. *Bulgarian Academy of Sciences-Problems of Engineering Cybernetics and Robotics*, 69:52–61, 2018.

[12] David Borland and Russell M Taylor II. Rainbow color map (still) considered harmful. *IEEE Computer Architecture Letters*, 27(02):14–17, 2007.

[13] Danielle Albers Szafir. The good, the bad, and the biased: Five ways visualizations can mislead (and how to fix them). *Interactions*, 25(4):26–33, 2018.

[14] Safdar Zaman, Wolfgang Slany, and Gerald Steinbauer. Ros-based mapping, localization and autonomous navigation using a pioneer 3-dx robot and their relevant issues. In *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, pages 1–5. IEEE, 2011.

[15] Fitria Romadhona Quratul Aini, Agung Nugroho Jati, and Unang Sunarya. A study of monte carlo localization on robot operating system. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 1–6. IEEE, 2016.

[16] Matthew B Luebbers, Connor Brooks, Carl L Mueller, Daniel Szafir, and Bradley Hayes. Arc-lfd: Using augmented reality for interactive long-term robot skill maintenance via constrained learning from demonstration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3794–3800. IEEE, 2021.

[17] Connor Brooks and Daniel Szafir. Visualization of intended assistance for acceptance of shared control. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11425–11430. IEEE, 2020.

[18] Toby Collett and Bruce A. MacDonald. An augmented reality debugging system for mobile robot software engineers. *Journal of Software Engineering for Robotics*, 1:18–32, 2010.

[19] DWF Van Krevelen and Ronald Poelman. A survey of augmented reality technologies, applications and limitations. *International journal of virtual reality*, 9(2):1–20, 2010.

[20] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001.

[21] John P McIntire, Paul R Havig, and Eric E Geiselman. What is 3d good for? a review of human performance on stereoscopic 3d displays. In *Head-and Helmet-Mounted Displays XVII; and Display Technologies and Applications for Defense, Security, and Avionics VI*, volume 8383, page 83830X. International Society for Optics and Photonics, 2012.

[22] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE transactions on visualization and computer graphics*, 18(12):2431–2440, 2012.

[23] Michael Kölling David J. Barnes. *Objects First with Java: A Practical Introduction Using BlueJ 6*. Objects First with Java: A Practical Introduction Using BlueJ 6th Edition. Pearson, 6th edition, 2016.

[24] Joseph Lawrance, Christopher Bogart, Margaret Burnett, Rachel Bellamy, Kyle Rector, and Scott D. Fleming. How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, 39(2):197–215, 2013.

[25] Thomas D LaToza and Brad A Myers. Developers ask reachability questions. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 185–194, 2010.

[26] Mark Weiser. Program slicing. *IEEE Transactions on software engineering*, (4):352–357, 1984.

[27] Andrew Ko and Brad Myers. Debugging reinvented. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 301–310. IEEE, 2008.

[28] Jun Kato, Takeo Igarashi, and Masataka Goto. Programming with examples to develop data-intensive user interfaces. *Computer*, 49(7):34–42, 2016.

[29] Daniel Szafir and Danielle Albers Szafir. Connecting human-robot interaction and data visualization. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 281–292, 2021.

[30] Daniel Szafir. Mediating human-robot interactions with virtual, augmented, and mixed reality. In *International Conference on Human-Computer Interaction*, pages 124–149. Springer, 2019.

[31] Zhanat Makhataeva and Huseyin Atakan Varol. Augmented reality for robotics: a review. *Robotics*, 9(2):21, 2020.

[32] Alan G. Millard, Richard Redpath, Alistair M. Jewers, Charlotte Arndt, Russell Joyce, James A. Hilder, Liam J. McDaid, and David M. Halliday. Ardebug: An augmented reality tool for analysing and debugging swarm robotic systems. *Frontiers in Robotics and AI*, 5:87, 2018.

[33] Sunao Hashimoto, Akihiko Ishida, Masahiko Inami, and Takeo Igarashi. Touchme: An augmented reality interface for remote robot control. *J. Robotics Mechatronics*, 25(3):529–537, 2013.

[34] Stephanie Arévalo Arboleda, Tim Dierks, Franziska Rücker, and Jens Gerken. There's more than meets the eye: Enhancing robot control through augmented visual cues. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 104–106, 2020.

[35] Kyeong-Beom Park, Sung Ho Choi, Jae Yeol Lee, Yalda Ghasemi, Mustafa Mohammed, and Heejin Jeong. Hands-free human–robot interaction using multimodal gestures and deep learning in wearable mixed reality. *IEEE Access*, 9:55448–55464, 2021.

[36] Hangxin Liu, Yaofang Zhang, Wenwen Si, Xu Xie, Yixin Zhu, and Song-Chun Zhu. Interactive robot knowledge patching using augmented reality. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1947–1954. IEEE, 2018.

[37] Mark Cheli, Jivko Sinapov, Ethan E. Danahy, and Chris Rogers. Towards an augmented reality framework for k-12 robotics education. *In Proceedings of the 1st International Workshop on Virtual, Augmented,and Mixed Reality for HRI*, 2018.

[38] Stéphane Magnenat, Morderchai Ben-Ari, Severin Klinger, and Robert W. Sumner. Enhancing robot programming with visual feedback and augmented reality. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, page 153–158, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Andre Cleaver, Faizan Muhammad, Amel Hassan, Elaine Short, and Jivko Sinapov. Sensar: A visual tool for intelligent robots for collaborative human-robot interaction, 2020.

[40] F. Muhammad, A. Hassan, A. Cleaver, and J. Sinapov. Creating a shared reality with robots. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 614–615, 2019.

[41] Jakob Nielsen. Ten usability heuristics.

[42] Edward Clarkson and Ronald C Arkin. Applying heuristic evaluation to human-robot interaction systems. In *Flairs Conference*, pages 44–49, 2007.

[43] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4. McLean, VA, USA, 2005.

[44] Martin Bischoff, David Whitney, and Eric Vollenweider. ros-sharp. https://github.com/EricVoll/ros-sharp.git, 2017.

[45] Tamara Munzner. *Visualization analysis and design*. CRC press, 2014.

[46] Virginia Braun and Victoria Clarke. Thematic analysis. 2012.

[47] Daniel M Russell, Mark J Stefik, Peter Pirolli, and Stuart K Card. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 269–276, 1993.

[48] Jared A Frank, Matthew Moorhead, and Vikram Kapila. Mobile mixed-reality interfaces that enhance human–robot interaction in shared spaces. *Frontiers in Robotics and AI*, 4:20, 2017.

[49] Dennis Sprute, Klaus Tönnies, and Matthias König. A study on different user interfaces for teaching virtual borders to mobile robots. *International Journal of Social Robotics*, 11(3):373–388, 2019.

[50] Bang Wong. Color blindness. *nature methods*, 8(6):441–442, 2011.

[51] Alex T Pang, Craig M Wittenbrink, Suresh K Lodha, et al. Approaches to uncertainty visualization. *The Visual Computer*, 13(8):370–390, 1997.

[52] Julian Kardos, Antoni Moore, and George Benwell. Expressing attribute uncertainty in spatial data using blinking regions. In *Proceedings of the 7th International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences, Lisbon, Portugal*, pages 5–7. Citeseer, 2006.

[53] Katherine Haramundanis. Why icons cannot stand alone. *ACM SIGDOC Asterisk Journal of Computer Documentation*, 20(2):1–8, 1996.

[54] Ruth Rosenholtz, Yuanzhen Li, and Lisa Nakano. Measuring visual clutter. *Journal of vision*, 7(2):17–17, 2007.

[55] Zoya Bylinskii, Nam Wook Kim, Peter O'Donovan, Sami Alsheikh, Spandan Madan, Hanspeter Pfister, Fredo Durand, Bryan Russell, and Aaron Hertzmann. Learning visual importance for graphic designs and data visualizations. In *Proceedings of the 30th Annual ACM symposium on user interface software and technology*, pages 57–69, 2017.

[56] Christopher Healey and James Enns. Attention and visual memory in visualization and computer graphics. *IEEE transactions on visualization and computer graphics*, 18(7):1170–1188, 2011.

[57] Fred Paas, Juhani E Tuovinen, Huib Tabbers, and Pascal WM Van Gerven. Cognitive load measurement as a means to advance cognitive load theory. *Educational psychologist*, 38(1):63–71, 2003.

[58] Yu-Tzu Lin, Cheng-Chih Wu, Ting-Yun Hou, Yu-Chih Lin, Fang-Ying Yang, and Chia-Hu Chang. Tracking students' cognitive processes during program debugging—an eye-movement approach. *IEEE transactions on education*, 59(3):175–186, 2015.

[59] Jessie YC Chen, Ellen C Haas, and Michael J Barnes. Human performance issues and user interface design for teleoperated robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1231–1245, 2007.

[60] M Alejandra Menchaca-Brandan, Andrew M Liu, Charles M Oman, and Alan Natapoff. Influence of perspective-taking and mental rotation abilities in space teleoperation. In *Proceedings of the ACM/IEEE International Conference on Human-robot interaction*, pages 271–278, 2007.

[61] Matt McFarland. Darpa's robotics challenge has a gender problem, Apr 2019.