ImageInThat: Manipulating Images to Convey User Instructions to Robots

***Abstract*—Foundation models are rapidly improving the capability of robots in performing everyday tasks autonomously such as meal preparation, yet robots will still need to be instructed by humans due to model performance, the difficulty of capturing user preferences, and the need for user agency. Robots can be instructed using various methods—natural language conveys immediate instructions but can be abstract or ambiguous, whereas end-user programming supports longer-horizon tasks but interfaces face difficulties in capturing user intent. In this work, we propose using direct manipulation of images as an alternative paradigm to instruct robots, and introduce a specific instantiation called *ImageInThat* which allows users to perform direct manipulation on images in a timeline-style interface to generate robot instructions. Through a user study, we demonstrate the efficacy of *ImageInThat* to instruct robots in kitchen manipulation tasks, comparing it to a text-based natural language instruction method. The results show that participants were faster with *ImageInThat* and preferred to use it over the text-based method.**

***Index Terms*—end-user robot programming, direct manipulation, robot instruction following**

I. Introduction

Advances in foundation models are rapidly improving the capabilities of autonomous robots, bringing us closer to robots entering our homes where they can complete everyday tasks. However, the need for human *instructions* will persist—whether due to limitations in robot policies, models trained on internet-scale data that may not capture the specifics of users' environments or preferences, or simply the desire for users to maintain control over their robots' actions. For instance, a robot asked to wash dishes might follow a standard cleaning routine—*e.g.*, by placing everything in the dishwasher and then putting them away in the cupboard—but may not respect a user's preferences—*e.g.*, needing to wash delicate glasses "by hand" or organizing cleaned dishes in a specific way—thus necessitating human intervention.

Existing methods for instructing robots range from those that focus on *commanding* the robot for the purpose of immediate execution (*e.g.*, uttering a language instruction to wash glasses by hand [1]) to methods that *program* the robot such as learning from demonstration [2] or end-user robot programming [3]. However, prior methods, whether they are used for commanding or programming, have notable drawbacks. Using language to command a robot can be quick and generate an immediate execution from the robot, although language can be abstract and difficult to ground in the robot's environment. For instance, the command *"put away the dishes in the cabinets after cleaning them"* is ambiguous if there are dishes of different types, each needing to be placed in different areas of the same cabinet or in entirely different cabinets. End-user programming promotes the creation of reusable programs for repeated execution, yet it remains challenging to capture user intents [3].

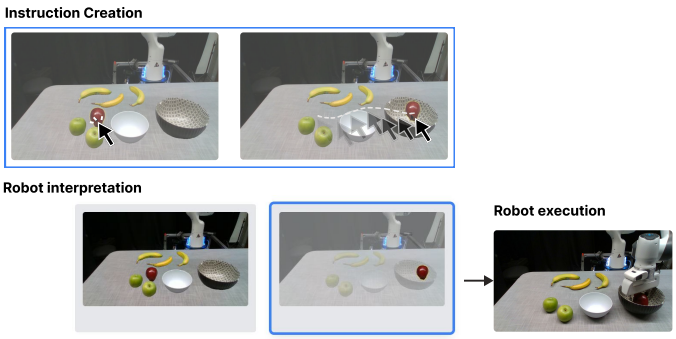

Fig. 1. We introduce the direct manipulation of images as a paradigm for providing instructions to a robot. Depicted in the bottom left are a series of instructions that a user is giving the robot by manipulating the fruits. The top shows one trajectory of direct manipulation

In this paper, we propose directly manipulating images on a visual interface as a means of instructing a robot. By *images*, we mean a visual observation of the robot's environment captured by a camera attached to the robot or the environment. Similar to how cleaning robots today present a visual map of their surroundings, we envision future robots offering access to image observations of the robot's environment. Compared to prior methods for instructing robots, images are easy to interpret, even for longer horizon tasks, since they are already grounded in the robot's environment. Moreover, direct manipulation [4] inherently reduces ambiguity, as actions such as manipulating an object within the image eliminate the need for descriptive instructions [5]. We introduce a specific instantiation of this paradigm, *ImageInThat*, which combines many state-of-the-art foundation models, and enables users to manipulate images of the robot's environment to create instructions. *ImageInThat* integrates several techniques:

- *Direct manipulation* of objects and fixtures to create instructions;
- A *timeline interface* for ordering instructions and assessing whether the shown changes achieve them;
- *Highlighting changes* between instructions using visualization methods to help users interpret changes;
- *Language-based image editing* to leverage the strengths of language to visualize instructions;
- *Automatic captioning* of user manipulations with text to

enhance confidence about the robot's understanding of instructions;
- *Predicting and proposing future instructions* based on contextual interpretation of user actions.

In a user study with ten participants, we compared *ImageInThat* to a language-based method for four instruction-following tasks in simulated kitchen environments. We found that participants were able to generate instructions in these tasks faster (64.8% less time) when using *ImageInThat*, and participants were more confident that their instructions could be understood by a robot when they directly manipulated images to provide instructions. We further demonstrate that image-based instructions created using *ImageInThat* can be translated and executed on a physical robot arm through a case study. This work advances the state-of-the-art by making the following contributions:

- An alternative paradigm of instructing robots enabled by the direct manipulation of images on a visual interface;
- An instantiation of this paradigm, *ImageInThat*, realized through a novel prototype;
- Results of a user evaluation comparing *ImageInThat* to an instance of a language-based method;
- Early demonstrations that user-generated images can be translated into robot actions.

II. Background

Prior robotics literature has proposed several paradigms using which humans can instruct robots. Broadly, these can be categorized as *commands*–instructions that are provided and executed in the moment, and *programs*–structured instructions that can be repeated in an automated fashion. Regardless of whether a robot is instructed through commands or a program, instructions require four major components: (1) capturing user intent, (2) translating that intent into a command or program, (3) presenting the command or program to the user for confirmation or feedback, and (4) executing the instruction.

Continuous commands. Commands can be issued in real-time or near real-time (referred to as control in the robotics literature) and involve continuous input from the user through *teleoperation*, for which several methods exist. Since continuous commands are typically executed in real-time, there is often no explicit notion of presenting user intent for confirmation or feedback (3), For instance, when using a physical device for teleoperation [6], [7], manipulating the device captures user intent to provide specific changes (deltas) to the robot's end effector pose or joint configuration, whereas execution occurs immediately through low-level controllers [8], [9]. The user gets feedback implicitly but in real time while the robot executes the command—by either observing the robot after executing a command physically [10] or visually during remote teleoperation [11]–[13]. In contrast, graphical user interfaces often provide the user feedback prior to execution, such as by visualizing valid commands [14]–[16], virtual surrogates that can be manipulated instead of the real robot [17]–[19], or visualizing and manipulating a 3D rendering of the robot's environment in 3D [20], [21].

Discrete commands. Other methods of issuing commands provide a layer of abstraction to reduce the user burden of having to provide continuous input. For instance, users can create instructions through natural language which are executed immediately [1], [22], [23]. Although this abstraction provides the opportunity for user feedback, it is often not utilized and the user interface is simply the language used (*e.g.,* natural language). Some recent work provides the opportunity for user intervention via language as an interface, such as where the robot engages in a dialogue to clarify instruction ambiguities [24], requests user help [25]. or lets the user provide iterative language corrections [26], [27]. From an execution standpoint, discrete commands such as those that are language-based can be executed by pretrained general-purpose large language models (LLMs) that translate them into policy code [28]–[31] or in an end-to-end fashion using trained special-purpose robot foundation models that map language to robot actions [1], [32]–[34].

Programs. Prior work has proposed methods to create instructions for longer-horizon tasks in a repeatable manner, and employ a variety of instruction representations. Graphical programming representations, such as blocks [35]–[37] and nodes [38], [39], are often used in end-user robot programming (EURP) systems. Other approaches utilize augmented reality [18], [40]–[43], sketching [44], [45], and tangibles [46]–[48]. Recently, LLMs have been employed to enable the use of freeform natural language to create text-based programs using chat interfaces [49], [50]. The creation step of a program involves defining the step-by-step procedure of the robot's behavior, for instance by dragging-and-dropping individual blocks in a visual editor [35].

Since EURP is typically done offline, most systems enable the user to provide confirmation to the system and gather feedback using a given representation. For instance, blocks in a program can be viewed in a visual editor whereas other methods contextualize the program such as with augmented visualizations on a tabletop [47], [51] or mapping the robot's environment within the editor [52]. This helps track the robot's future states and identify possible errors. Beyond viewing and confirmation steps, most EURP systems support editing instructions natively. Programs are often executed using classical methods such as motion planning [47] or program synthesis [53]. However, LLMs have also recently been employed to generate code from a program [49].

Our work does not distinctively fit into either paradigm, but borrows from and has applicability to both. We also utilize an instruction representation, namely, images, which has been less explored for instructing robots, particularly for the end user. Using images as a representation offers several benefits. Instructions can be created quickly through direct manipulation, is concrete by nature (since manipulation occurs on a specific object of interest), and reduces ambiguities that are inherent to other methods (*e.g.*, language). This approach can be used to *command* a robot in near real time (akin to language). Further, because images are concrete, they can help the user in tracking the robot's state over time. Hence, it can

support the creation of procedures for longer-horizon tasks as with *programming*. Lastly, using images as a representation supports flexible execution, such as by translating images to robot policy code [28] (which we demonstrate in this work), and potentially translating images into language captions for executing trough language-conditioned policies [34], or immediately used via image-conditioned policies [33].

III. Image Manipulation through ImageInThat

We introduce manipulating images as a new way to instruct robots. Here, we describe the interactions that a user can have with *ImageInThat*, a prototype instantiation of this concept. To enable these interactions, *ImageInThat* assumes that there will be a setup phase where the robot builds an internal representation of the user's environment in which it will operate. This representation consists of knowledge about objects and fixtures. *Objects* are items that can be manipulated from one location to another (*e.g.*, bowls that need washing) whereas *fixtures* are items that are immovable. Both objects and fixtures can be in more than one state; for example, a cabinet is a fixture that can be open or closed. Throughout our description of the *ImageInThat* system, we will use an example of how a user might have their robot take a particular bowl out of a cabinet that they wish to use to hold an apple by utilizing *ImageInThat*. To support the creation of new instructions, *ImageInThat* includes an editor.

Direct manipulation to interact with objects and fixtures (Figure 2A. In the *ImageInThat* interface, the user can perform simple drag-and-drop operations in the editor to manipulate their locations upon selecting them. The user can also change the order in which objects appear by right clicking them. They can toggle the state of fixtures by clicking on them if they take on a discrete number of states. Here we use the example of instructing the robot to take a bowl from a cabinet to show how the user can manipulate objects and fixtures to construct commands. This task can be decomposed this into three *steps*: (1) open the cabinet; (2) choose a bowl, and move it from the cabinet to the counter; and (3) close the cabinet. In *ImageInThat*, specifying each of these steps is achieved by performing clicking to toggle between the fixture's states (opening and closing the cabinet) and drag-and-drop operations (moving the bowl). This concreteness is in contrast to prior methods for instructing robots that can inherently contain ambiguities.

Timeline to interpret the continually changing environment (Figure 2B). *ImageInThat* provides a visual timeline to help the user anchor and understand their instructions to the robot in the context of the continually changing environment. This is especially useful for longer-horizon tasks. When the user opens the interface, the initial state of the environment is prepopulated in the timeline. This serves as the starting point for all instructions for the robot, and future steps represent changes from this initial step. Each step in the timeline is displayed from left to right (temporally) as small thumbnail images. When the user selects a step, it becomes populated inside the editor and available to be modified. Each time the user toggles the state of a fixture by clicking it, a new step is inserted after the current step in the timeline. In a similar manner, every time the user performs a drag-and-drop manipulation on a new object, the timeline is populated with a new step. Continuously manipulating the same object allows the user to refine the object's position without creating unnecessary steps. Sometimes, a set of instructions might involve the same object moving in the environment, such as when a dirty bowl needs to be taken from the counter and rinsed under running water before being placed inside a sink. To support creating such instructions, the user can copy a step from the timeline by selecting a button that appears under the step. They can also delete steps using a button under the step in the timeline. The timeline highlights another benefit of images—the ability to visually track how the robot will manipulate the environment to perform instructions rather than having to imagine it.

Visually highlighting changes. Since images are information dense, and the timeline can hold many images at a time, naïvely placing images in the timeline may not help the user visually track their instructions over time. To assist with this, *ImageInThat* utilizes two visualizations. First, each time a step is populated in the timeline, *ImageInThat*'s internal representation already maintains a log of *changes* between consecutive steps. When an object is moved between consecutive steps, it is visually highlighted while all other objects and fixtures are made less salient (Figure 2C). In contrast, when a fixture's state changes, the environment is made more visible (where the fixture is located) by keeping it at full opacity and making all objects less visible. This allows users to quickly scan the timeline and gauge what has changed at each step. When a step is selected, hovering over any previous or future step displays the difference through an animation, showing any changes in object positions and fixture states.

Blending language and image. Image manipulation has the benefit of being concrete. For instance, the manipulation of individual objects may cause them to move to a new location. As an example, the user may wish to put an apple into the bowl they took out of the cabinet. However, since direct manipulation only lets the user move the apple on top of the bowl (as there is no way for it to be put "inside" without simulating physics), the user may be unsure whether the robot has understood their instruction. *ImageInThat* automatically annotates each step using a *caption* (Figure 2D). Captions describe the change between the current step and the previous step. A caption for moving the apple might read, "`Move the apple from the cabinet into the bowl`." By default, an image and its corresponding caption within the same step are linked together. Hence, any changes to the caption will also modify the image. The user can also modify a caption to provide additional context about the intent of their manipulation to the robot by unlinking the caption from the corresponding image within the step. For instance, the user might want to specify that apples should only be inside the bowl after they have been washed.

There are also many situations where the user has a higher-

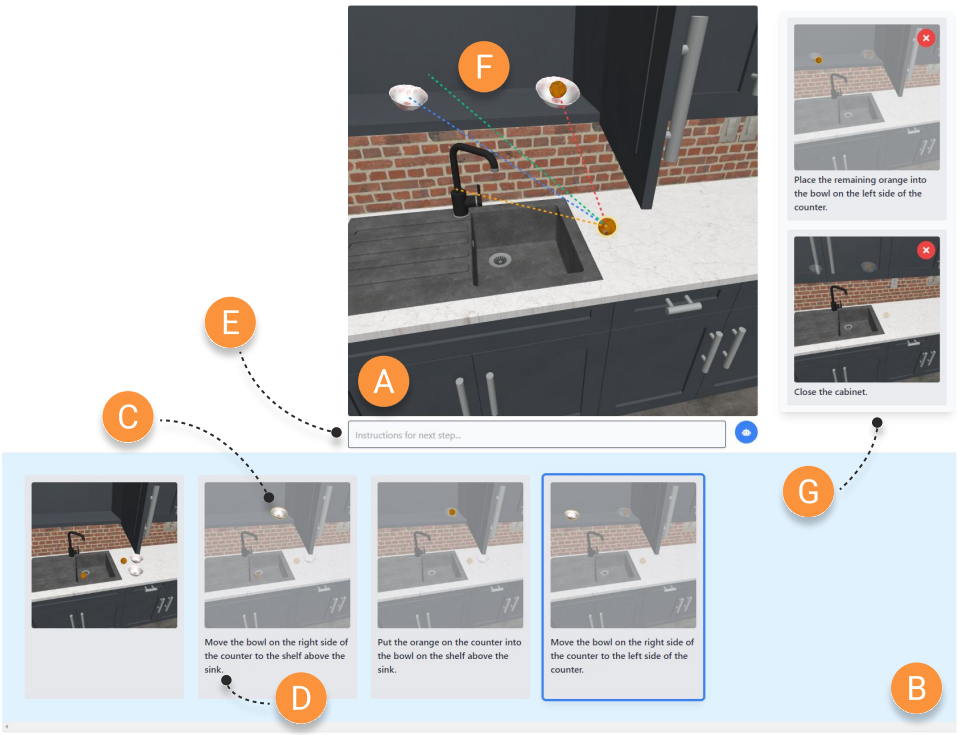

Fig. 2. *ImageInThat*'s user interface, consisting of an editor (top) and a timeline (bottom). The editor allows users to manipulate objects and fixtures in the environment, while the timeline displays the current state of the environment and the desired changes. The timeline (B) shows all instructions to the robot. Selecting a step populates it in the editor (B). Changes between steps are made visible by contrasting changed objects and fixtures from other items (C). *ImageInThat* automatically captions all manipulations and allows them to be edited. The user can instruct the robot with text to generate new steps automatically. *ImageInThat* tries to predict user goals such as by proposing locations where objects can be placed (F) or predicting a future step (G).

level goal (*e.g.*, doing the dishes) but either does not know the sequence of steps needed to achieve the goal or does not want to specify them by hand. *ImageInThat* allows the user to input a language instruction when a step is selected and populated in the editor (Figure 2E). Depending on the specificity of the instruction, one or many steps, each containing an image describing the change, are automatically generated and populated in the timeline.

Predicting user goals. Guided by the initial setup phase, *ImageInThat* determines the locations of all objects and fixtures. It uses this knowledge to propose goal locations for a selected object (Figure 2F). For instance, when the user selects the apple, *ImageInThat* could propose placing it inside the sink (for washing before use) or inside the bowl (for storing). These choices are visualized as lines originating from the selected object. Selecting any part of the line performs the manipulation for the user instead of requiring a drag-and-drop operation. *ImageInThat* keeps track of all steps that are populated in the timeline. It uses the contextual information embedded in the timeline to predict what the user might want to do next, and proposes these as plausible next steps outside the timeline (Figure 2) that the user can choose between (*e.g.*, two options). Selecting a plausible step adds it to the timeline while allowing the user to reject one or all plausible steps.

IV. Implementation

High-level system design. *ImageInThat* consists of two major components: (1) a *back-end server* that manages machine learning models to generate the image representations for user manipulation and enables features such as automatic captioning, and (2) a *user interface* for viewing and editing the images, enabling users to create robot instructions. The server is realized as a Flask application that enables two-way communication between the models and the interface. The models are hosted as TCP sockets on workstations within the local network and communicate with the Flask server to respond to requests from the interface. The user interface is built using ReactJS. Our implementation for the user study utilizes images from kitchen environments created using Robocasa [54] whereas the real-world deployment uses streamed camera images. *ImageInThat* uses GPT-4o (gpt-4o-2024-05-13) as the large-language-model (LLM) for producing captions, enabling language-based image edits, and predicting user goals, though this can be swapped for other foundation models. Please see the appendices for more details.

ImageInThat creates an initial representation of the robot's environment by extracting information about the objects and fixtures. *ImageInThat* assumes access to a predefined list of plausible objects and fixtures plus their states (*e.g.*, a drawer being able to open or close). While the information about object and fixture is predefined in our implementation, it could

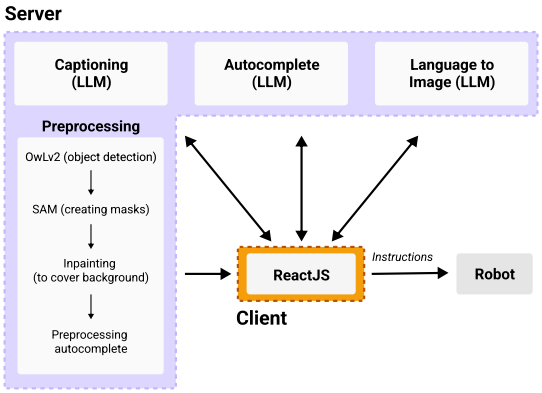

Fig. 3. System diagram of *ImageInThat* showing its major components. The server side handles the preprocessing step and all intelligent features that require interfacing with the LLM (*e.g.*, autocomplete, captioning, and language to step generation. The client is a web user interface built with ReactJS.

also be inferred through automated methods that detect object presences, states, and possible articulation (e.g. [55]–[57]).

Creating the initial representation for user manipulation. In *ImageInThat*, fixture states are realized as background images while object masks are overlaid on top of them. Each combination of states is used to generate an image representing the environment in that state. For example, in a kitchen environment with a drawer and a cabinet, one possible state is the drawer being open while the cabinet remains closed. In the user study, the images are generated by API calls to Robocasa to modify the fixtures aided by generated code from the LLM whilst hiding all objects. We argue that this is a reasonable assumption given that many existing robot applications assume the existence of digital twins [58]. We also experimented with fine-tuning language-conditioned diffusion models [59], [60] to generate images where fixtures change state using a language prompt for environments that do not have a digital twin. For these environments, the parts of the background behind the objects are reconstructed using inpainting techniques [61].

To enable the direct manipulation of objects, *ImageInThat* must identify and create masks for them. To achieve this, the LLM is prompted to produce a list of visible items in the environment from the list of plausible ones. Then, an open vocabulary object detector (OwLv2 [62]) finds the corresponding bounding boxes for both objects and fixtures. Object bounding boxes are then processed using Segment Anything [63] to generate masks. For detected fixtures, *ImageInThat* generates interactable regions using the bounding boxes produced by the object detector, enabling mouse clicks within these regions to activate state changes.

Initializing the environment. After generating backgrounds as well as representing objects and fixtures, the server transmits an environment object and the initial state to the user interface. The environment object represents all static aspects of the environment, including all objects plus their bounding boxes, fixtures with their bounding boxes and their possible states, and the backgrounds corresponding to all fixture state combinations. The initial state, on the other hand, describes the initial location of all objects (*i.e.*, x and y position) and the states of all fixtures. The user interface renders the initial state of the environment as a step in the timeline represented as an SVG image. In this initial state, object order is also determined using the LLM, whereby receptacles (*e.g.*, bowls) are behind other objects (*e.g.*, fruits). Once the environment is populated, the user can interact with the environment by manipulating objects and fixtures. Each time the user interacts with the environment, new environment states are created (or existing states are updated) to keep track of the locations and states of the fixtures, which subsequently adds new visual steps to the timeline. Changes between steps are displayed by applying filters to the difference between their environment states.

Blending language and image. Each time a new step is created through a drag-and-drop manipulation, the LLM is prompted with data about the environment, environment states at each step, and the corresponding images to generate a caption. For fixture state changes, captions are created simply by appending the object type and the state.

When the user enters a language instruction while a step is selected, the data about the environment, the corresponding environment state, and the corresponding image are used to prompt the LLM to classify the instruction as either requiring a fixture state change or an object manipulation. Depending on the result, a subsequent call is made to the LLM to either produce an updated list of fixture states, or manipulate the 2D coordinates of the corresponding object(s). Editing a step's caption produces an updated step using the same approach. By default, just a single step is created with the user's instruction. *ImageInThat* also supports adding more abstract instructions (*e.g.*, wash the dishes) that can be further broken down into smaller instructions, each producing a step.

Predicting user goals. During pre-processing, the LLM is prompted to filter objects in the environment that can be manipulated by the robot, and all the locations they could be placed. This is used to propose goal locations when the user selects an object. Lastly, to propose plausible steps, the user can press a button near the editor to prompt the LLM with the environment plus all environment states corresponding to all steps until the selected step as well as the corresponding images, with the goal of generating any number of plausible actions (a system parameter) based on the robot's existing skills and the sequence of steps in the timeline. Each action is used to generate a plausible step by determining whether it requires a fixture state change or an object manipulation.

V. Evaluation

We evaluated *ImageInThat* through a user study conducted in a laboratory with ten participants recruited using university and professional networks $M = 25.9$ years, $SD = 3.38$ years (5 women, 5 men) who rated their familiarity with providing instructions to a robot on a seven-point Likert scale ($M = 4.1$, $SD = 2.1$). In the study, we compared an instance of *ImageInThat* with a language-based method.

Fig. 4. Tasks performed by participants in the evaluation (left to right): *organizing pantry*, *sorting fruits*, *cooking stir-fry*, and *washing dishes*. Depicted here is the environment state at the *beginning* of each task.

Conditions. To allow a comparison of each modality, we omitted all the language features of *ImageInThat*, including the ability to generate new images using language instructions or modify them using captions. For the language condition, we re-purposed *ImageInThat*, replacing all image-based interactions with text. Instead of populating images in the timeline, the user populates the timeline with textual descriptions of task steps. In both conditions, participants provided instructions pertaining to one object at a time, reflecting the current capabilities of robots, which typically limited to single-object manipulation. Further, most language-conditioned robot policies (*e.g.*, [34]) follow a similar approach, executing single language instructions at a time. While LLMs can interpret more abstract instructions and decompose them, they may make mistakes that require corrections [26].

Study design and tasks. The study utilizes a within-subjects design with two conditions–image and text–that are presented in a counterbalanced order to minimize ordering effects. Within each condition, participants completed four tasks where they instructed a robot to complete kitchen manipulation tasks. The tasks ranged in difficulty from easy to hard: *storing pantry*, *sorting fruits*, *cooking stir-fry*, and *washing dishes*. The tasks where chosen to assess different types of instruction following including specifying: individual objects when there may be duplicates (*e.g.*, an apple on the counter versus one inside a bowl), spatial constraints when placing objects (*e.g.*, placing an onion to the left of a big potato), dealing with occluded objects (*e.g.*, removing a large oil bottle which is blocking the olive oil bottle that needs to be used), and lastly, keeping track of object locations and context as they undergo various manipulations (*e.g.*, taking food off of a dirty plate, washing it, and storing it in the cabinet). Within each condition, the four tasks were assigned in random order. After both condition blocks, participants completed a freeform task to experiment with the features that were excluded from *ImageInThat*. Details of the individual tasks can be found in the Appendices and Supplementary Material.

Measures. In the study, we collected data on participants' performance when using both methods. Quantitative measures include task completion time and number of errors, which were determined by comparison to an *oracle* representation of the task established *a priori* by two researchers. An error was recorded if: a participant missed a step included in the oracle; there was an extraneous step that was not seen in the oracle; or if a step from the oracle was inefficiently broken into multiple steps by the participant. We also measured subjective perceptions of the prototypes, including participants' confidence in correctly communicating their intent to the robot, workload through the NASA TLX questionnaire [64], and system usability (with the System Usability Scale which includes 10 items on a five-point scale [65]).

Procedure. Participants first provided consent and completed a pre-study questionnaire assessing their familiarity with robots and instructing them. After watching a video tutorial introducing *ImageInThat* and the text-based method and brief experimentation with both, participants began one of the two study condition blocks. Between tasks, participants rated how confident they were that the robot could understand their instructions unambiguously on a seven-point Likert scale. At the end of each condition block, two questionnaires (NASA TLX and SUS) were administered to assess workload and usability, respectively. At the end of the study, participants rated their preference for the text-based method compared to *ImageInThat* on a seven-point Likert scale. Lastly, we conducted a brief interview probing participants about various aspects of both methods.

Hypotheses. We formulated three hypotheses: Participants will complete tasks faster using the *ImageInThat* compared to the text-based method (**H1**); Participants will make fewer errors using the *ImageInThat* compared to the text-based method (**H2**); Participants will feel more confident that a robot unambiguously understands their instructions when using *ImageInThat* compared to the text-based method (**H3**). To test these hypotheses we used a paired t-test for all metrics to account for repreated measures.

Findings from Measures of Performance. Figure 5 provides a breakdown of participants' completion time by condition and task. Participants were faster ($t(37) = -8.96, p < 0.001$) with *ImageInThat* ($M = 110.8$ seconds, $SD = 70.04$ seconds) compared to the text-based method ($M = 363.88$ seconds, $SD = 186.45$ seconds).

Findings from Measures of Error.. Overall there was no significant difference in errors ($t(37) = -0.76, p > 0.05$), however *ImageInThat* ($M = 2.26$, $SD = 3.07$) had slightly fewer errors than the text-based method ($M = 2.55$, $SD = 3.02$). The breakdown of errors suggests the text-based method

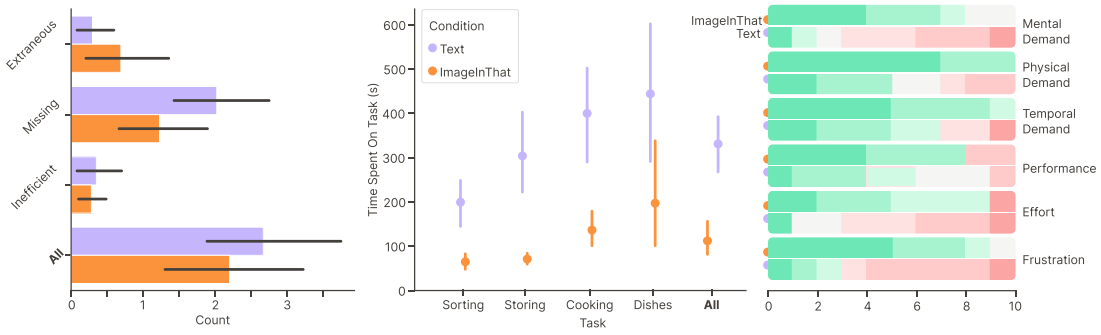

Fig. 5. Left: a plot showing the number of errors for both *ImageInThat* and the Text system. Errors are broken into the three categories extraneous steps, missing steps, and inefficient steps. A bar also displays the total count of all errors. The middle plot shows the task completion time for all 4 tasks, and the completion time across all tasks. Finally at right are the counts of responses for the NASA-TLX questionnaire. All error bars are bootstrapped 95% confidence intervals.

had more ($t(37) = -2.43, p < 0.05$) missing steps ($M = 2.0$, $SD = 2.11$) than *ImageInThat* ($M = 1.26$, $SD = 1.98$). However, there was no difference ($t(37) = 1.35, p > 0.05$) in extraneous steps between *ImageInThat* () and the text-based method. There was also no significant difference between the text-based method also had fewer inefficient steps ($M = 0.38$, $SD = 0.92$) than *ImageInThat* ($M = 0.31$, $SD = 0.61$). **Findings from Measures of Error.**. Overall there was no significant difference in errors ($t(37) = -0.76, p > 0.05$), however *ImageInThat* ($M = 2.26$, $SD = 3.07$) had slightly fewer errors than the text-based method ($M = 2.55$, $SD = 3.02$). The breakdown of errors suggests the text-based method had more ($t(37) = -2.43, p < 0.05$) missing steps ($M = 2.0$, $SD = 2.11$) than *ImageInThat* ($M = 1.26$, $SD = 1.98$). However, there was no difference ($t(37) = 1.35, p > 0.05$) in extraneous steps between *ImageInThat* (($M = 0.29$, $SD = 0.61$)) and the text-based method ($M = 0.26$, $SD = 0.79$). There was also no significant difference ($t(37) = 1.35, p > 0.05$) between the text-based method ($M = 0.26$, $SD = 0.79$) and *ImageInThat* ($M = 0.29$, $SD = 0.61$) with respect to inefficient steps.

Findings from Subjective Measures. Participants felt more confident ($t(37) = 5.63, p < 0.001$) that their instructions would be understood unambiguously by a robot when using the *ImageInThat* ($M = 6.42$, $SD = 0.92$) versus the text-based method ($M = 4.71$, $SD = 1.75$). When comparing system usability, *ImageInThat* received a higher ($t(9) = 5.75, p < 0.001$) average score ($M = 87.75$, $SD = 14.16$) than the text-based method ($M = 61.75$, $SD = 23.07$). Lastly, participants reported a lower workload on the NASA TLX when using *ImageInThat* compared to the text-based method (Figure 5). Based on these scores, *ImageInThat* can be interpreted as having "excellent" usability while the text-based mehod would be classified as having "marginal" usability.

VI. Translating *ImageInThat* to Robot Actions

While images may help users specify instructions more easily, they must be possible to execute by the robot. There are potentially various methods to translate images to robot actions, For instance, it could be possible to generate robot policy code [28] by processing user-generated images rather than text which could be executed on the robot. We did a preliminary assessment of translation from images to code. To achieve this, we prompt the LLM ten times. As execution performance of the programs are tightly coupled with the robot's skills and environment model (e.g., point cloud quality), we do not assess this. We defined a set of APIs for the robot to use including: *pick*, *place*, *grasp*, and *ungrasp*.

To generate policy code, we prompt an LLM (gpt-4o-2024-05-13) with the predefined APIs, images corresponding to the sequence of steps, the environment, and the environment states (from *ImageInThat*). We attempted to generate code for two tasks, one simple and two of a medium difficulty both involving sorting fruits. The easy task has two steps requiring the robot to put a red apple in the white bowl (there are two green apples and a red apple on the table). In the medium difficulty task, the green apples should both put in the white bowl while a red apple should be put in the patterned bowl.

In the task requiring the robot to put the red apple into the white bowl, the translation was successful 10/10 times. Similarly, the first medium difficulty task involving sorting the apples by bowl type (green apples in the white bowl and red apples in the black bowl), the translation was successful.

Although we did not conduct a thorough assessment of the execution of the code on the robot, we built a pipeline to achieve this using skill primitives (defined earlier). The input parameter to the pick primitive is used to find the relevant environmental object using an open vocabulary object detector (OwLv2 [62]). This is used to generate an object point cloud by segmenting its 3D bounding box [63], after which sample candidate grasps using Contact-GraspNet [66]. The place primitive also takes a string description of an object and uses the detector to find and returns a pose for the robot to drop off items.

VII. Discussion

Contextualizing the performance of *ImageInThat*. The results of our user evaluation suggest that *ImageInThat* led to faster completion of the instruction giving tasks with fewer errors. Specifically, participants had more missing steps when they used the text-based method. One rationale could be that

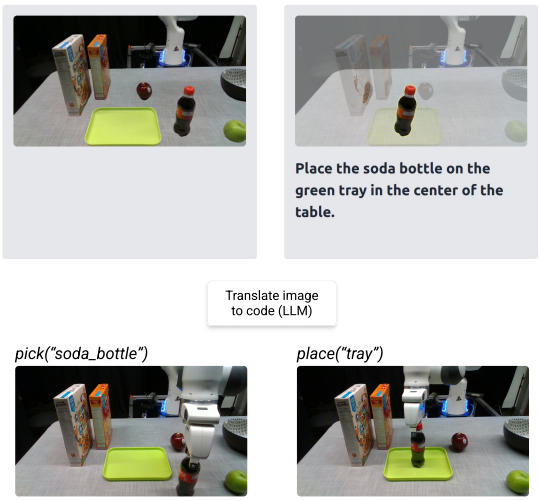

Fig. 6. Figure showing the Franka Panda executing code generated by translating images generated with the *ImageInThat* interface.

images support keeping track of the environment state over longer-horizon tasks so they were less likely to miss a step, However, participants included more extraneous steps (i.e., steps that do not contribute to the goal) possibly due to the low cost of adding new instructions afforded by direct manipulation.

Several factors that could lead to participants spending significantly more time and effort giving instructions in language. We notice that the additional cost of using the text-only modality were more pronounced in resolving ambiguities, for example, when referencing specific objects or achieving precise object placement. P9 commented "*The text interface was incredibly cumbersome...Because the room for ambiguity in the instructions made it so that each step required a lot of mental processing to try to remove any ambiguities in my instruction.*" Participants often needed to use complex referring expressions when trying to disambiguate between multiple similar objects, whereas with *ImageInThat* they could select the target object directly. In a similar vein, participants often had to construct complex sentences to describe the exact locations for objects to be placed. Apart from these two types of instructions, participants also reported challenges in error correction with text, as it requires to "*create a series (of new instructions)*" (P4).

Participants noted the extraneous cognitive effort for making sense of multi-step plans as they get longer in the text-based condition. They talked about the difficulty in mentally tracking object location and state changes, and how it negatively impacted their confidence in their instructions. Further, they liked the immediate visual feedback that *ImageInThat* affords to help them evaluate action feasibility and plan further, contrasting it with the text-based condition where "*I have to imagine what the result of this (text-based) is going to be... it's like playing a game of blindfolded chess.*" (P6).

Limitations. Though we are encouraged by the findings that participants made fewer errors, there are some caveats. First, our comparison required participants to provide step-by-step instructions for a robot when using the text-based method. While this is how robots act on instructions, humans may think at higher abstraction levels. Here, there is the potential for LLMs to take a higher-level instruction and break it down into a sequence of lower-level instructions for the robot, though these are prone to errors [26]. Hence, we note that the performance of *ImageInThat* represents the best-case scenario for the image manipulation paradigm and the worst-case scenario for the text-based method (Figure 5). Technical limitations also bound *ImageInThat*'s performance, including that of the underlying models for detecting objects reliably, masking them into meaningful objects, and the ability for vision language models to perceive environment changes. In particular, 2D editing could sometimes be affected by the problem of perspective. When objects are manipulated, they tend to change size and appear out of place which may impact detection or recognition performance.

Future Work. Although there is some early evidence that user-manipulated images could be translated into robot code, it is preliminary. It would be important to thoroughly assess where image manipulation can succeed and where it has shortcomings. For instance, we noticed (anecdotally) that vision language models find it challenging to describe small changes (e.g., when a bowl shifts a bit to one direction or another). Further, we would like to assess how robot foundation models—particularly those that condition on goal images would perform if provided user-manipulated images since these would likely be out of their training distribution (as image manipulation inherently leads to awkward object scaling and perspective). Lastly, it would be interesting to explore whether interaction traces when using systems like *ImageInThat* can be leveraged to learn user patterns and preferences for tasks. For instance, observing the user put heavier dishes on lower shelves could prompt similar suggestions or better initial proposed steps from the system.

VIII. Conclusion

In this work, we proposed the direct manipulation of images as a new paradigm for robot instruction, and showed its feasibility and capabilities through our prototype system, *ImageInThat*. Through user studies comparing *ImageInThat* with a text-based baseline across four complex kitchen manipulation tasks, we have shown that image manipulation enables the quick specification of robot instructions with less effort and workload. Through this work, we also took a step towards future interfaces that blend image and language seamlessly as each modality has their benefits (such as being able to be abstract or repeat an instruction many times with language). This multimodal form of instruction could benefit users with differing capabilities and needs.We hope that this inspires significant future efforts towards methods that support humans in instructing robots.

REFERENCES

[1] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," **IEEE Robotics and Automation Letters**, 2023.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," **Robotics and autonomous systems**, vol. 57, no. 5, pp. 469–483, 2009.

[3] G. Ajaykumar, M. Steele, and C.-M. Huang, "A survey on end-user robot programming," **ACM Computing Surveys (CSUR)**, vol. 54, no. 8, pp. 1–36, 2021.

[4] B. Shneiderman, "Direct manipulation: A step beyond programming languages," **Computer**, vol. 16, no. 08, pp. 57–69, 1983.

[5] D. Masson, S. Malacria, G. Casiez, and D. Vogel, "Directgpt: A direct manipulation interface to interact with large language models," in **Proceedings of the CHI Conference on Human Factors in Computing Systems**, 2024, pp. 1–16.

[6] D. Gopinath, S. Jain, and B. D. Argall, "Human-in-the-loop optimization of shared autonomy in assistive robotics," **IEEE robotics and automation letters**, vol. 2, no. 1, pp. 247–254, 2016.

[7] R. Temma, K. Takashima, K. Fujita, K. Sueda, and Y. Kitamura, "Third-person piloting: Increasing situational awareness using a spatially coupled second drone," in **Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology**, 2019, pp. 507–519.

[8] K. Darvish, L. Penco, J. Ramos, R. Cisneros, J. Pratt, E. Yoshida, S. Ivaldi, and D. Pucci, "Teleoperation of humanoid robots: A survey," **IEEE Transactions on Robotics**, vol. 39, no. 3, pp. 1706–1727, 2023.

[9] D. J. Rea and S. H. Seo, "Still not solved: A call for renewed focus on user-centered teleoperation interfaces," **Frontiers in Robotics and AI**, vol. 9, p. 704225, 2022.

[10] D. Rakita, B. Mutlu, and M. Gleicher, "Effects of onset latency and robot speed delays on mimicry-control teleoperation," in **HRI'20: Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction**, 2020.

[11] D. Wei, B. Huang, and Q. Li, "Multi-view merging for robot teleoperation with virtual reality," **IEEE Robotics and Automation Letters**, vol. 6, no. 4, pp. 8537–8544, 2021.

[12] A. Naceri, D. Mazzanti, J. Bimbo, D. Prattichizzo, D. G. Caldwell, L. S. Mattos, and N. Deshpande, "Towards a virtual reality interface for remote robotic teleoperation," in **2019 19th International Conference on Advanced Robotics (ICAR)**. IEEE, 2019, pp. 284–289.

[13] D. Rakita, B. Mutlu, and M. Gleicher, "An autonomous dynamic camera method for effective remote teleoperation," in **Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction**, 2018, pp. 325–333.

[14] D. Kent, C. Saldanha, and S. Chernova, "A comparison of remote robot teleoperation interfaces for general object manipulation," in **Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction**, 2017, pp. 371–379.

[15] J. Li, R. Balakrishnan, and T. Grossman, "Starhopper: A touch interface for remote object-centric drone navigation," 2020.

[16] E. Rosen, D. Whitney, E. Phillips, G. Chien, J. Tompkin, G. Konidaris, and S. Tellex, "Communicating robot arm motion intent through mixed reality head-mounted displays," in **Robotics research: The 18th international symposium ISRR**. Springer, 2020, pp. 301–316.

[17] M. E. Walker, H. Hedayati, and D. Szafir, "Robot teleoperation with augmented reality virtual surrogates," in **2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)**. IEEE, 2019, pp. 202–210.

[18] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. M. Van der Loos, and E. Croft, "Robot programming through augmented trajectories in augmented reality," in **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. IEEE, 2018, pp. 1838–1844.

[19] K. Mahadevan, Y. Chen, M. Cakmak, A. Tang, and T. Grossman, "Mimic: In-situ recording and re-use of demonstrations to support robot teleoperation," in **Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology**, 2022, pp. 1–13.

[20] Y. Li, S. Agrawal, J.-S. Liu, S. K. Feiner, and S. Song, "Scene editing as teleoperation: A case study in 6dof kit assembly," in **2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. IEEE, 2022, pp. 4773–4780.

[21] S. Aoyama, J.-S. Liu, P. Wang, S. Jain, X. Wang, J. Xu, S. Song, B. Tversky, and S. Feiner, "Asynchronously assigning, monitoring, and managing assembly goals in virtual reality for high-level robot teleoperation," in **2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)**. IEEE, 2024, pp. 450–460.

[22] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in **Experimental robotics: the 13th international symposium on experimental robotics**. Springer, 2013, pp. 403–415.

[23] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," **Annual Review of Control, Robotics, and Autonomous Systems**, vol. 3, no. 1, pp. 25–55, 2020.

[24] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar **et al.**, "Inner monologue: Embodied reasoning through planning with language models," **arXiv preprint arXiv:2207.05608**, 2022.

[25] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley **et al.**, "Robots that ask for help: Uncertainty alignment for large language model planners," **arXiv preprint arXiv:2307.01928**, 2023.

[26] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh, "Distilling and retrieving generalizable knowledge for robot manipulation via language corrections," **arXiv preprint arXiv:2311.10678**, 2023.

[27] J. Liang, F. Xia, W. Yu, A. Zeng, M. G. Arenas, M. Attarian, M. Bauza, M. Bennice, A. Bewley, A. Dostmohamed **et al.**, "Learning to learn faster from human feedback with language model predictive control," **arXiv preprint arXiv:2402.11450**, 2024.

[28] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in **2023 IEEE International Conference on Robotics and Automation (ICRA)**. IEEE, 2023, pp. 9493–9500.

[29] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in **2023 IEEE International Conference on Robotics and Automation (ICRA)**. IEEE, 2023, pp. 11 523–11 530.

[30] P. Liu, Y. Orru, C. Paxton, N. M. M. Shafiullah, and L. Pinto, "Ok-robot: What really matters in integrating open-knowledge models for robotics," **arXiv preprint arXiv:2401.12202**, 2024.

[31] K. Mahadevan, J. Chien, N. Brown, Z. Xu, C. Parada, F. Xia, A. Zeng, L. Takayama, and D. Sadigh, "Generative expressive robot behaviors using large language models," in **Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction**, 2024, pp. 482–491.

[32] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu **et al.**, "Rt-1: Robotics transformer for real-world control at scale," **arXiv preprint arXiv:2212.06817**, 2022.

[33] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu **et al.**, "Octo: An open-source generalist robot policy," **arXiv preprint arXiv:2405.12213**, 2024.

[34] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi **et al.**, "Openvla: An open-source vision-language-action model," **arXiv preprint arXiv:2406.09246**, 2024.

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

[36] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in **2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)**. IEEE, 2016, pp. 295–302.

[37] 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.

[38] 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.

[39] D. Porfirio, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Authoring and verifying human-robot interactions," in **Proceedings of the 31st annual acm symposium on user interface software and technology**, 2018, pp. 75–86.

[40] B. Ikeda and D. Szafir, "Programar: Augmented reality end-user robot programming," **ACM Transactions on Human-Robot Interaction**, vol. 13, no. 1, pp. 1–20, 2024.

[41] R. Suzuki, A. Karim, T. Xia, H. Hedayati, and N. Marquardt, "Augmented reality and robotics: A survey and taxonomy for ar-enhanced human-robot interaction and robotic interfaces," in **Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems**, 2022, pp. 1–33.

[42] L. Gong, S. Ong, and A. Nee, "Projection-based augmented reality interface for robot grasping tasks," in **Proceedings of the 2019 4th International Conference on Robotics, Control and Automation**, 2019, pp. 100–104.

[43] Y. Cao, T. Wang, X. Qian, P. S. Rao, M. Wadhawan, K. Huo, and K. Ramani, "Ghostar: A time-space editor for embodied authoring of human-robot collaborative task with augmented reality," in **Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology**, 2019, pp. 521–534.

[44] D. Sakamoto, K. Honda, M. Inami, and T. Igarashi, "Sketch and run: a stroke-based interface for home robots," in **Proceedings of the SIGCHI conference on human factors in computing systems**, 2009, pp. 197–200.

[45] D. Porfirio, L. Stegner, M. Cakmak, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Sketching robot programs on the fly," in **Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction**, 2023, pp. 584–593.

[46] Y. S. Sefidgar, P. Agarwal, and M. Cakmak, "Situated tangible robot programming," in **Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction**, 2017, pp. 473–482.

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

[48] D. J. Porfirio, L. Stegner, M. Cakmak, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Figaro: A tabletop authoring environment for human-robot interaction," in **Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems**, 2021, pp. 1–15.

[49] U. B. Karli, J.-T. Chen, V. N. Antony, and C.-M. Huang, "Alchemist: Llm-aided end-user development of robot applications," in **Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction**, 2024, pp. 361–370.

[50] Y. Ge, Y. Dai, R. Shan, K. Li, Y. Hu, and X. Sun, "Cocobo: Exploring large language models as the engine for end-user robot programming," **arXiv preprint arXiv:2407.20712**, 2024.

[51] Y. S. Sefidgar, T. Weng, H. Harvey, S. Elliott, and M. Cakmak, "Robotist: Interactive situated tangible robot programming," in **Proceedings of the 2018 ACM Symposium on Spatial User Interaction**, 2018, pp. 141–149.

[52] G. Huang, P. S. Rao, M.-H. Wu, X. Qian, S. Y. Nof, K. Ramani, and A. J. Quinn, "Vipo: Spatial-visual programming with functions for robot-iot workflows," in **Proceedings of the 2020 CHI conference on human factors in computing systems**, 2020, pp. 1–13.

[53] D. Porfirio, E. Fisher, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Bodystorming human-robot interactions," in **proceedings of the 32nd annual ACM symposium on user Interface software and technology**, 2019, pp. 479–491.

[54] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," **arXiv preprint arXiv:2406.02523**, 2024.

[55] J. Kerr, C. M. Kim, M. Wu, B. Yi, Q. Wang, K. Goldberg, and A. Kanazawa, "Robot see robot do: Imitating articulated object manipulation with monocular 4d reconstruction," **arXiv preprint arXiv:2409.18121**, 2024.

[56] J.-B. Alayrac, I. Laptev, J. Sivic, and S. Lacoste-Julien, "Joint discovery of object states and manipulation actions," in **Proceedings of the IEEE International Conference on Computer Vision**, 2017, pp. 2127–2136.

[57] S. Qian, L. Jin, C. Rockwell, S. Chen, and D. F. Fouhey, "Understanding 3d object articulation in internet videos," in **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 2022, pp. 1599–1609.

[58] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani **et al.**, "Evaluating real-world robot manipulation policies in simulation," **arXiv preprint arXiv:2405.05941**, 2024.

[59] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," in **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 2023, pp. 18 392–18 402.

[60] K. Black, M. Nakamoto, P. Atreya, H. Walke, C. Finn, A. Kumar, and S. Levine, "Zero-shot robotic manipulation with pretrained image-editing diffusion models," **arXiv preprint arXiv:2310.10639**, 2023.

[61] O. Elharrouss, N. Almaadeed, S. Al-Maadeed, and Y. Akbari, "Image inpainting: A review," **Neural Processing Letters**, vol. 51, pp. 2007–2028, 2020.

[62] M. Minderer, A. Gritsenko, and N. Houlsby, "Scaling open-vocabulary object detection," **Advances in Neural Information Processing Systems**, vol. 36, 2024.

[63] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo **et al.**, "Segment anything," in **Proceedings of the IEEE/CVF International Conference on Computer Vision**, 2023, pp. 4015–4026.

[64] S. G. Hart, "Nasa-task load index (nasa-tlx); 20 years later," in **Proceedings of the human factors and ergonomics society annual meeting**, vol. 50, no. 9. Sage publications Sage CA: Los Angeles, CA, 2006, pp. 904–908.

[65] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," **Intl. Journal of Human–Computer Interaction**, vol. 24, no. 6, pp. 574–594, 2008.

[66] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes," in **2021 IEEE International Conference on Robotics and Automation (ICRA)**. IEEE, 2021, pp. 13 438–13 444.