ImageInThat: Manipulating Images to Convey User Instructions to Robots

Karthik Mahadevan
Department of Computer Science
University of Toronto
Toronto, Canada
karthikm@dgp.toronto.edu

Blaine Lewis
Department of Computer Science
University of Toronto
Toronto, Canada
blaine@dgp.toronto.edu

Jiannan Li
School of Computing & Information Systems
Singapore Management University
Singapore, Singapore
jiannanli@smu.edu.sg

Bilge Mutlu
Department of Computer Sciences
University of Wisconsin-Madison
Madison, USA
bilge@cs.wisc.edu

Anthony Tang
School of Computing & Information Systems
Singapore Management University
Singapore, Singapore
tonyt@smu.edu.sg

Tovi Grossman
Department of Computer Science
University of Toronto
Toronto, Canada
tovi@dgp.toronto.edu

***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. Supplementary material including code can be found at: https://image-in-that.github.io/.**

***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.

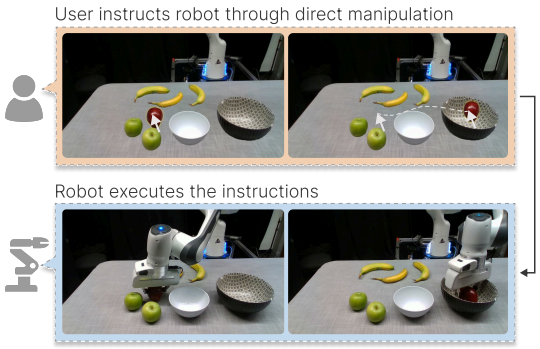

Fig. 1. We introduce a new paradigm for instructing robots through the direct manipulation of images. *ImageInThat* is a specific instantiation of this paradigm where users can manipulate images in a timeline-style interface to create instructions for the robot to execute.

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. In contrast, end-user programming promotes the creation of reusable programs for repeated execution, yet it remains challenging to capture user intents [3].

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. Akin to how cleaning robots today present a 2D 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. By enabling the creation of intuitive, image-based instructions, *ImageInThat* addresses a key HRI challenge [6]: facilitating effective information exchange between humans and robots. Our method allows users to directly manipulate images to convey intent, with the robot implicitly communicating its understanding of the user's intent by enabling these image manipulations. 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 [7], [8], 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 [9], [10]. 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 [11] or visually during remote teleoperation [12]–[14]. In contrast, graphical user interfaces often provide the user feedback prior to execution, such as by visualizing valid commands [15]–[17], virtual surrogates that can be manipulated instead of the real robot [18]–[20], or visualizing and manipulating a 3D rendering of the robot's environment in 3D [21], [22].

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], [23], [24]. 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 [25], requests user help [26], or lets the user provide iterative language corrections [27], [28]. 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 [29]–[32] or in an end-to-end fashion using trained special-purpose robot foundation models that map language to robot actions [1], [33]–[35].

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 [36]–[38] and nodes [39], [40], are often used in end-user robot programming (EURP) systems. Other approaches utilize augmented real-

ity [19], [41]–[44], sketching [45], [46], and tangibles [47]–[49]. Recently, LLMs have been employed to enable the use of freeform natural language to create text-based programs using chat interfaces [50], [51]. 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 [36].

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 [48], [52] or mapping the robot's environment within the editor [53]. 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 [48] or program synthesis [54]. However, LLMs have also recently been employed to generate code from a program [50].

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 [29] (which we demonstrate in this work), and potentially translating images into language captions for execution through language-conditioned policies [35], or immediately used via goal image-conditioned policies [34].

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 put an orange inside a bowl and put

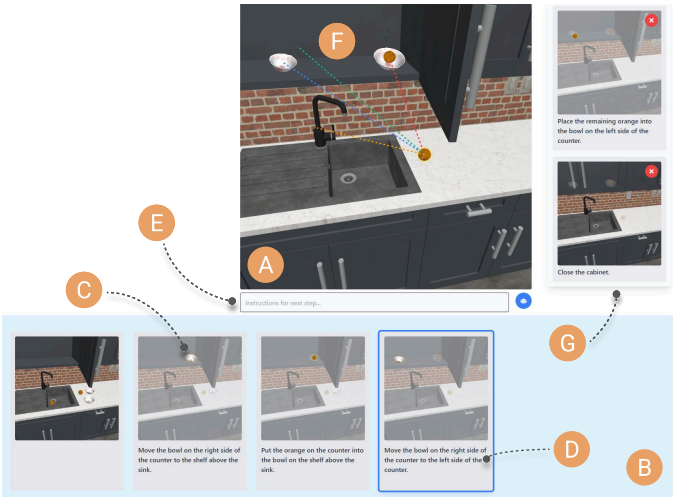

Fig. 2. *ImageInThat*'s user interface, consisting of an editor (top) and a timeline (bottom). The editor (A) allows users to manipulate objects and fixtures in the environment, while the timeline displays the current state of the environment and the desired changes. The timeline (B) shows all instructions provided to the robot. Selecting a step populates it in the editor. 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 (D). The user can instruct the robot with text to generate new steps automatically (E). *ImageInThat* tries to predict user goals such as by proposing locations where objects can be placed (F) or proposing future steps (G).

it inside the cabinet 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. In the example of putting a bowl with an orange inside it in the cabinet, the task can be decomposed this into three *steps*: (1) put the orange in a bowl; (2) place the bowl inside the cabinet; 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 pre-populated 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. The timeline highlights another benefit of images—the ability to visually track how the robot will manipulate the environment to perform instructions rather than imagining 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 this, *ImageInThat* utilizes two visualizations. First, each time a step is populated in the timeline, *ImageInThat*'s internal representation 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 orange into the bowl. However, since direct manipulation only lets the user move the orange 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 orange might read, "`Move the orange from the counter into the bowl`." By default, an image and its corresponding caption within the same step are linked together. Hence, any changes manually made to the caption will also modify the image.

There are also many situations where the user has a higher-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 orange, *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, and proposes these as plausible next steps (Figure 2G) 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. Two versions of *ImageInThat* were developed: a simulated (**Sim**) version for the user study and a version to demonstrate real-world usage (**Real**). *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 a workstation (with an Nvidia RTX A6000) within the local network and communicate with the Flask server to respond to requests from the interface. The user interface is built using ReactJS. To populate the interface, **Sim** utilizes images from kitchen environments created using Robocasa [55] whereas **Real** 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 (see Appendices). *ImageInThat* creates an initial representation of the robot's environment by extracting information about the objects and fixtures. **Sim** assumes access to a predefined list of plausible objects and fixtures whereas **Real** prompts the LLM with the robot's observation to generate the list of plausible objects and fixtures. However, fixture states are predefined (*e.g.*, a drawer being able to open or close) though they could be inferred through automated methods (*e.g.,* [56]–[58]).

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.

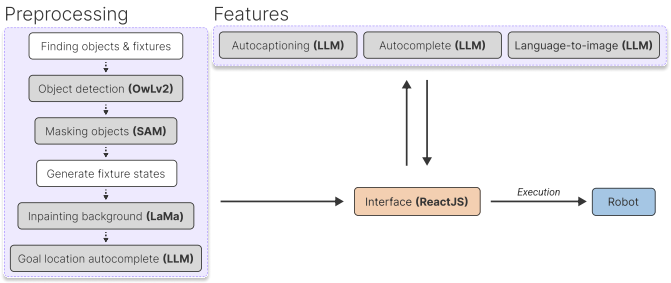

Fig. 3. System diagram of *ImageInThat* showing its major components. The server side (in purple) handles the preprocessing step and all intelligent features that require interfacing with the LLM (*e.g.*, autocomplete, captioning, and language-to-step generation). The client is a web user interface built with ReactJS. Components in white are implemented differently for the user study and real-world usage.

In **Sim**, the background images are generated by API calls to Robocasa to modify the fixtures aided by generated code from the LLM whilst hiding all objects whereas in **Real**, images representing different states are captured a priori. We argue that this is a reasonable assumption given that many existing robot applications assume the existence of digital twins [59]. We also experimented with fine-tuning language-conditioned diffusion models [60], [61] to generate images where fixtures change state using a language prompt for environments that do not have a digital twin (Figure 4). For these environments, the parts of the background behind the objects are reconstructed using inpainting techniques [62].

To enable direct manipulation, *ImageInThat* must identify and create masks for objects. Using the list of plausible objects and fixtures, an open vocabulary object detector (OwLv2 [63]) finds the corresponding bounding boxes. Objects' bounding boxes are then processed using Segment Anything [64] to generate masks. In **Real**, inpainting is used to fill the background behind detected objects. 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. For the study (**Sim**), the fixtures' bounding boxes were pre-determined.

Initializing the environment. After generating the backgrounds and 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 robot's 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* 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 through an SVG image. In the initial state, object order is determined by prompting 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. When the user interacts with the environment, new environment states are created (or existing states are updated) to track 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 with 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 type and 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. Although the current implementation uses an LLM for manipulating items in each step, fine-tuned language condition diffusion models could also be used (*e.g.,* Figure 4).

Predicting user goals. During pre-processing, the LLM is prompted to filter which objects in the environment 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 as a form of *autocomplete*. 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 prompt is then used to generate a plausible step and shown to the user to select or reject; if selected, the step is added to the timeline.

System requirements and latency. In *ImageInThat*, hosting large models requires significant VRAM, with the fine-tuned diffusion model being the most demanding. Preprocessing can require minutes, including background generation per fixture state, object detection, and mask extraction. Inpainting and determining object goal locations requires a few seconds due to LLM calls and modifying the background image per mask respectively. After preprocessing, interactions occur in real-time. Automatic captioning runs in the background, while image editing through language and proposing plausible steps require longer (tens of seconds), involving multiple LLM calls.

V. Evaluation

We evaluated *ImageInThat* through a user study conducted in a laboratory with ten participants recruited using university

Fig. 4. Sampled results of providing a language instruction to a fine-tuned language-conditioned diffusion model to modify fixture states. In each pair of images, the left represents the initial image and the right represents the generated image.

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.

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 is typically limited to single-object manipulation. Further, most language-conditioned robot policies (*e.g.*, [35]) follow a similar approach, executing single language instructions at a time.

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*. Further details can be found in the Appendices.

Fig. 5. 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.

Measures. In the study, we collected data on participants' performance when using both methods. Quantitative measures include task completion time and number of errors. An *oracle* representation of each task was created by two experimenters *a priori* representing the best possible performance on the task (with the correct instructions) and used to compare participants' performance. 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 [65], and system usability (with the System Usability Scale which includes 10 items on a five-point scale [66]). Due to data corruption, one task from Participant 2 and one task from Participant 7 were omitted from the analysis,

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 briefly experimenting 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 *ImageInThat* compared to the text-based method (**H1**); Participants will make fewer errors using *ImageInThat* compared to the text-based method (**H2**); and 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 repeated measures.

Findings from measures of performance. Figure 6 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

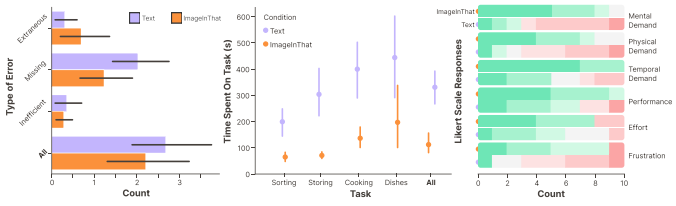

Fig. 6. Left: a plot showing the number of errors for *ImageInThat* and the text-based method. Errors are grouped into three categories: extraneous steps, missing steps, and inefficient steps, and a bar displays the total count of all errors. Middle: shows the task completion time per task and all tasks together. All error bars are bootstrapped 95% confidence intervals. Right: shows counts of responses for the NASA-TLX questionnaire.

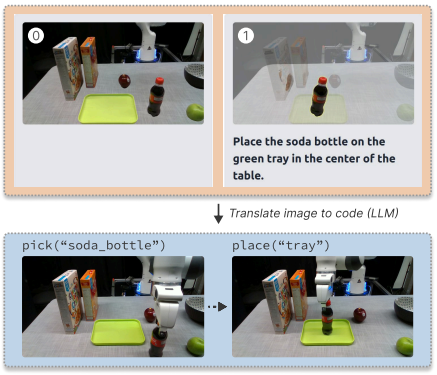

Fig. 7. We attempted to translate the images generated using *ImageInThat* to execute the underlying instructions. One such method for translation is from image to code that utilizes skill primitives by prompting an LLM. Illustrated here is one execution of the instructions.

no significant difference in errors ($t(37) = -0.76, p > 0.05$); however using *ImageInThat* ($M = 2.26$, $SD = 3.07$) led to 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 ($M = 61.75$, $SD = 23.07$). Based on these scores, *ImageInThat* can be interpreted as having "excellent" usability while the text-based method would be classified as having "marginal" usability. Lastly, participants reported a lower workload on the NASA TLX when using *ImageInThat* compared to the text-based method (Figure 6).

VI. Translating Images to Robot Actions

Once the user has specified their instructions through the direct manipulation of images, the robot must be able to execute them, This could be accomplished in several ways. In this work, we illustrate a case study of translating images into policy code (akin to *code as policies* [29] but with both text and image). We attempted image-to-code translation with 4 tasks. In our assessment, we provide a few pre-defined skill primitives as code APIs to prompt an LLM (gpt-4o-2024-05-13): *pick*, *place*, *grasp*, *ungrasp*, *turn on faucet*, *turn off faucet*, and *stack object*. In these tasks, we prompt the LLM ten times and focus on translation performance, as execution performance is influenced by factors like robot perception (*e.g.,* point cloud quality) and hardware limitations. The prompt to the LLM included: predefined APIs, images corresponding to the sequence of steps (without their associated captions), and data about the environment and the environment states (as processed by *ImageInThat*).

The first task (easy) required the robot to put a red apple into a white bowl in a table featuring a bowl, two green apples and a red apple (like Figure 1). Here, the translation was successful 10 out of 10 times. The second task (medium) required the robot to put both green applies into the white bowl and the red apple into the bowl with the pattern. This translation was also successful 10 out of 10 times. The third task (medium) required the robot to put a bowl in the sink, place an orange inside it, and wash the fruit by turning on the faucet (similar to the setup in Figure 2). This task was successful 8 out of 10 times, but in 6 runs the translation incorrectly moved the second orange into the second bowl as well though without moving it to the sink or washing it. In the fourth task (hard), we used part of the fourth evaluation task (Figure 5) whereby the robot needed to stack the orange donut on the plate containing the pink donut. The translation succeeded 7 out of 10 times, but in 3 runs, it mistakenly included code to rearrange spoons. In the latter two tasks, we noticed that the code often often defaulted to the "1st" item when there were duplicates (*e.g.,* "plate 1"), struggling to distinguish between similar objects, and extra code was often included based on incorrectly detecting state changes, highlighting the challenges VLMs face in handling many objects, duplicate objects, and subtle state changes.

Lastly, we illustrate through a case study that our approach can be realized to perform robot manipulation. In our pipeline, the *pick* primitive is executed by finding the desired object using OwLv2 [63], which is used to find a mask and create an object pointcloud [64]. Then, we sample candidate grasps using Contact-GraspNet [67]. The *place* primitive takes a string description of an object and uses OwLv2 to find and return a pose for the robot to drop off the item. Figure 7 shows a sample execution using this approach.

VII. Discussion

Contextualizing the performance of *ImageInThat*. The results of our user study 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 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 could have led to participants spending significantly more time and effort giving instructions in language. We noticed that the additional cost of using the text-only method were more pronounced in resolving ambiguities, such as when referencing specific objects or achieving precise object placement. P9 commented, *"The text interface was incredibly cumbersome. The room for ambiguity in the instructions made it so that each step required a lot of mental processing to remove any ambiguities in my instruction"*. Participants often used complex expressions to distinguish similar objects, whereas with *ImageInThat* they could manipulate the target object directly. Similarly, participants often constructed complex sentences to specify object placement. Participants also reported challenges in correcting errors once multiple steps were in the timeline, as it required them to *"create a series (of new instructions)."* (P4). Participants noted the increased cognitive effort in reasoning about and creating plans for longer tasks using text. Specific issues included the difficulty in mentally tracking object locations and state changes, which reduced their confidence in their instructions: *"I have to imagine what the result of this. It's like playing a game of blindfolded chess."* (P6). In contrast, participants appreciated the immediate visual feedback from *ImageInThat*.

Limitations and future work. Though we are encouraged by the findings that participants performed better with *ImageInThat* and preferred it, 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 decompose it into a sequence of lower-level instructions (*e.g.,* [27]) which could make using language less cumbersome. 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 6). Further, we represented language as text to enable comparisons via similar 2D user interfaces, but acknowledge that users may prefer speaking to the robot. Future work should compare speech to image-based methods. Our study findings are conditioned on a small sample size of ten participants. Future work should assess *ImageInThat* with a larger participant pool on more diverse, longer-horizon tasks beyond the kitchen domain. Lastly, while we demonstrate the multimodal capabilities of *ImageInThat*, our evaluation focused separately on text-based and image-based approaches, acknowledging the need to assess their combined effectiveness.

Technical limitations also bound *ImageInThat*'s performance particularly when deployed "in-the-wild" both for the creation of instructions and their execution. This includes detecting and masking objects reliably, especially when the environment is cluttered, and the ability for vision language models to perceive environment changes. For instance, when editing in 2D, the issues of object perspective could affect detection or recognition performance and require innovative solutions. For translating images into policy code, small changes can be difficult for vision language models to detect. There are also challenges with discriminating which objects are manipulated when there are identical objects.

Thus far, we focused on one-way interaction, *i.e.,* the initial provision of commands, but we can envision this being extended to enable back-and-forth interaction between the user and the robot such as for correcting manipulation errors. Further, it would be interesting to explore whether user interaction traces in *ImageInThat* could help learn user patterns and preferences, such as suggesting steps based on observed behaviors like placing heavier dishes on lower shelves. Lastly, we would like to assess other methods for translating image instructions to robot actions such as robot foundation models that condition on goal images when provided user-manipulated images that contain scaling and perspective artifacts, as these would likely fall outside their training distribution.

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, leveraging the strengths of each modality. This multimodal form of instruction could benefit users with differing capabilities and needs. We hope that this inspires significant future efforts into methods that support humans in instructing robots.

Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council of Canada IRCPJ-545100-18 grant, the National Science Foundation award IIS-1925043, and the Singapore Ministry of Education AcRF Tier 1 23-SIS-SMU-069 and 22-SIS-SMU-092 grants.

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] M. A. Goodrich, A. C. Schultz **et al.**, "Human–robot interaction: a survey," **Foundations and Trends® in Human–Computer Interaction**, vol. 1, no. 3, pp. 203–275, 2008.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J. Li, R. Balakrishnan, and T. Grossman, "Starhopper: A touch interface for remote object-centric drone navigation," in **Proceedings of the Graphics Interface Conference 2020**, 2020.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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," in **Proceedings of Machine Learning Research**, vol. 205, 2023, pp. 1769–1782.

[26] 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," in **Proceedings of the 7th Conference on Robot Learning**, 2023.

[27] 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," in **2024 IEEE International Conference on Robotics and Automation (ICRA)**, 2024.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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," in **Robotics: Science and Systems**, 2023.

[34] 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," in **Robotics: Science and Systems**, 2024.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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," in **IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)**, 2024.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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," in **8th Annual Conference on Robot Learning**, 2024.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

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

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

[64] 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.

[65] 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.

[66] 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.

[67] 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.