PATI: A Projection-Based Augmented Table-Top Interface for Robot Programming

Yuxiang Gao
Johns Hopkins University
Baltimore, Maryland
ygao73@jhu.edu

Chien-Ming Huang
Johns Hopkins University
Baltimore, Maryland
cmhuang@cs.jhu.edu

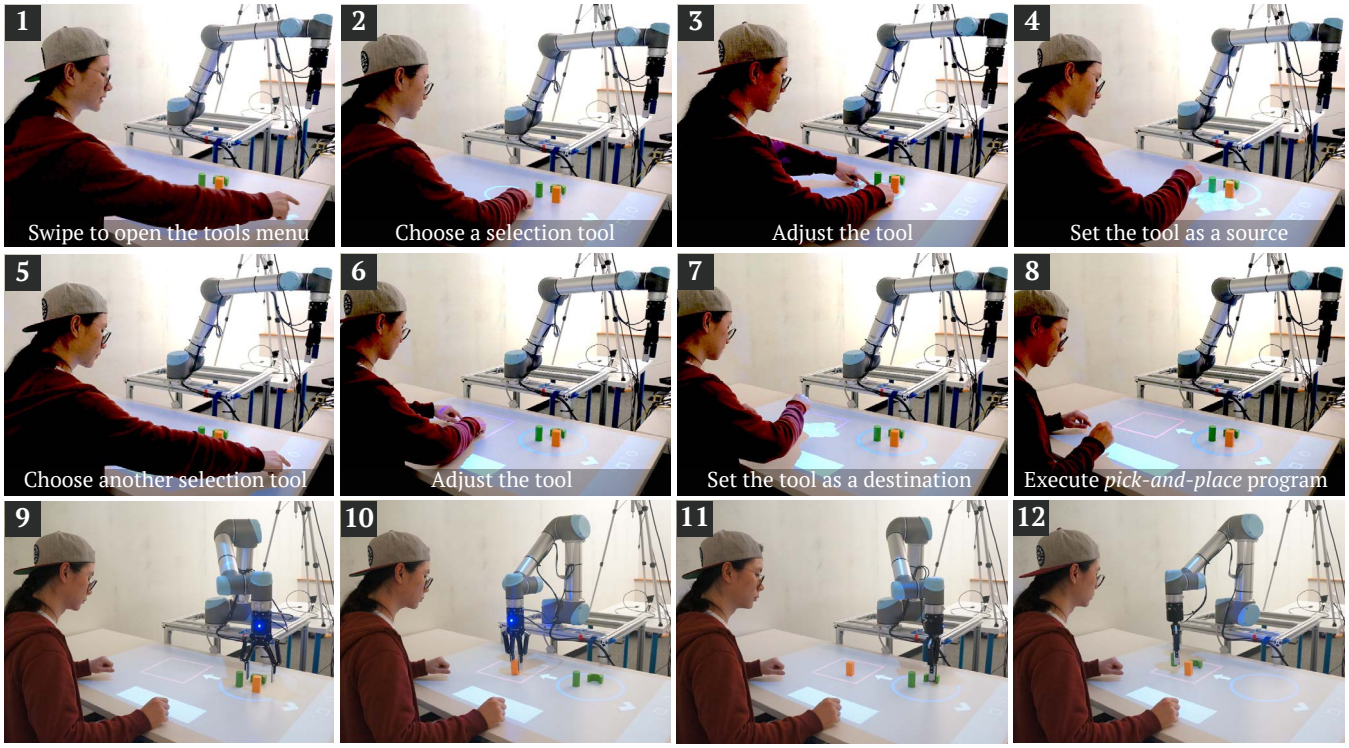


Figure 1: PATI supports the augmented and situated specification of robot tasks, allowing users to teach their robots new tasks by directly referencing and interacting with the environment through tools (e.g., shape tools) and common touch screen gestures, such as pinch, tap, or drag-and-drop. (1)–(8) show the process of specifying a pick-and-place task, and (9)–(12) show the robot performing the programmed task.

ABSTRACT

As robots begin to provide daily assistance to individuals in human environments, their end-users, who do not necessarily have substantial technical training or backgrounds in robotics or programming, will ultimately need to program and "re-task" their robots to perform a variety of custom tasks. In this work, we present PATI—a Projection-based Augmented Table-top Interface for robot programming—through which users are able to use simple, common gestures (e.g., pinch gestures) and tools (e.g., shape tools) to specify table-top manipulation tasks (e.g., pick-and-place) for a robot manipulator. PATI allows users to interact with the environment directly when providing task specifications; for example, users can utilize gestures and tools to annotate the environment with task-relevant information, such as specifying target landmarks and selecting objects of interest. We conducted a user study to compare PATI with a state-of-the-art, standard industrial method for end-user robot programming. Our results show that participants needed significantly less training time before they felt confident in using our system than they did for the industrial method. Moreover, participants were able to program a robot manipulator to

IUI '19, March 17–20, 2019, Marina del Rey, CA, USA

ACM ISBN 978-1-4503-6272-6/19/03…$15.00
https://doi.org/10.1145/3301275.3302326

complete a pick-and-place task significantly faster with PATI. This work indicates a new direction for end-user robot programming.

CCS CONCEPTS

• **Human-centered computing → Mixed / augmented reality**; **Graphical user interfaces**; **Collaborative interaction**; *Usability testing*; *Gestural input*; *User interface programming*; *Ubiquitous computing*; • **Computer systems organization → Robotics**; • **Computing methodologies →** Scene understanding; Vision for robotics.

KEYWORDS

Robot Programming, Human-Robot Interaction, End-User Programming, Projection-Based Interface, Tangible User Interface

ACM Reference Format:
Yuxiang Gao and Chien-Ming Huang. 2019. PATI: A Projection-Based Augmented Table-Top Interface for Robot Programming. In *24th International Conference on Intelligent User Interfaces (IUI '19), March 17–20, 2019, Marina del Rey, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3301275.3302326

1 INTRODUCTION

Robots have the potential to augment human abilities in a variety of domains, including flexible manufacturing, healthy aging, and collaborative construction. These domains necessitate that robots provide custom assistance to accommodate individual needs, environmental constraints, and task requirements. One way to enable robots to achieve such customization is to allow end-users to program and re-skill their robots according to the needs, constraints, and requirements presented in their own specific contexts. However, robot programming is difficult; it requires substantial technical training, knowledge in robotics and relevant engineering areas, and years of experience in order to become truly competent. To render robot programming more accessible to people lacking technical training or knowledge, we present a Projection-based Augmented Table-top Interface (PATI) for robot programming[1]. PATI is designed to let people use common gestures and tools to teach their robots new tasks with direct reference to and interactions with the environment in an augmented, situated manner (Figure 1).

Research in *Programming by Demonstration* (PbD) [4, 6, 8] has explored methods and interfaces for end-users to "program" new tasks for their robots through physical demonstrations, and seeks to generalize the learned skills for use in potential, unexplored scenarios. This body of research has investigated how people may train robot learners through kinesthetic demonstrations [1, 16], natural instructions with verbal and non-verbal behaviors [23, 24, 26, 33, 34], visual programming [2, 13, 18, 28], and teleoperative demonstrations in virtual reality [41]. However, common PbD methods involve little perception of the environment and rely on replaying demonstrated action trajectories or tracing recorded key waypoints (e.g., [1]).

While limited perception constrains the generalizability and scalability of PbD, perceiving task-relevant objects and the environment usually requires specialized processes for training objects for visual recognition and robot manipulation (e.g., [18, 28]), which consequently creates additional technical barriers for end-users. Moreover, referencing objects in the environment for robot manipulation is challenging due to the high variance in references, constrained perception, and different perspectives involved. Although prior research has explored methods for object referencing for robot manipulation [10, 27, 32, 35, 40], we still lack a simple, flexible—yet reliable—method for object referencing, which is essential for robots working alongside end-users.

In this paper, we seek to streamline the processes of visual tracking, object referencing, and task specification by enabling users to reference and annotate the environment directly through gestural input. To this end, we developed PATI, an experimental system designed to explore the possibilities and limitations of illustration-based robot programming. The design and development of PATI are motivated by prior research in end-user robot programming and projection-based interaction. In the next section, we review and discuss relevant prior work. We then present the concept of illustration-based robot programming and our implementation of the PATI system in Section 3. Section 4 describes a user study that evaluates the effectiveness of PATI and compares it with a state-of-the-art system of robot programming by demonstration. We report our results of this evaluation in Section 5, and conclude this paper with a discussion on the applications and limitations of PATI in Section 6.

2 BACKGROUND

2.1 End-User Robot Programming

The utility of a robot depends predominantly on its programmability [25]. To enable robotic aid in automation processes, daily human activities, and more, robotics research has explored various methods, tools, and interfaces for effective robot programming, all of which seek to teach and adapt robot skills to a variety of task configurations [5]. For example, the Robot Operating System (ROS) [29] has become a common programming framework for robotics research and industrial products. While offering a general framework for implementing algorithms and systems of sensing, planning, and decision-making, it requires significant training and experience before one can use it to competently program robot tasks. Moreover, it requires developers to have a wide span of knowledge—including mechanical engineering, control theory, and computer vision—in order to program robots to interact with people, objects, and the environment. Such technical barriers prevent laypeople from using robots in custom contexts and limit the possible benefits that robots are able to provide.

Programming by Demonstration (PbD) [4, 6, 8] is a robot programming paradigm that aims to lower such barriers by allowing end-users to program robots through action demonstration. Visual programming interfaces provide alternative methods of robot programming and seek to empower users with diverse backgrounds and levels of expertise (e.g., [2, 13, 18, 28]). While PbD is useful for specifying action configurations and trajectories relevant to task goals, visual programming abstracts low-level programming and allows users to focus on high-level task planning. A hybrid use of PbD and visual programming combines motion-level preciseness and task-level logics, and has become a common practice of robot programming in both industry (e.g., Universal Robots) and academia (e.g., [18, 28]). However, this hybrid approach still lacks

[1]This project is available at https://github.com/intuitivecomputing/PATI.

intuitive methods for referencing the objects and parts of the environment necessary for manipulation tasks—even those as simple as pick-and-place.

This paper aims to address these limitations by introducing a new method of robot programming, with which end-users will be able to specify robot tasks in situ through direct interaction with objects and the environment. The most relevant prior work is perhaps the framework of situated tangible robot programming [32]. This framework allows users to use tangible blocks to specify task-relevant objects and provide annotation in the environment. However, tangible programming does not support 3D specification (e.g., labeling the height of an object), and its expressiveness is limited by the number of blocks used. The work in this paper intends to explore an alternative method of situated programming enabled by projection-based augmentation.

2.2 Projection-Based Interactive Systems

In the domain of human-computer interaction, projection-based interfaces have been explored and used for ubiquitous access to and direct manipulation of information (e.g., [14, 20, 36, 37]). This type of interface requires minimal modification of the environment and allows users to interact directly with the existing environment. For example, everyday surfaces, such as the seat of a sofa or the surface of a wall, can be turned into augmented interfaces for interaction [37]. This unique quality makes projection-based interfaces particularly attractive for the seamless integration of novel technologies—including collaborative robots—into human environments with well-established infrastructures.

The use of projection-based interfaces has also been explored in the context of human-robot interaction (e.g., [3, 9, 12, 19, 31]). Such interfaces have mainly been used as a visualization method to display or convey spatial information in a situated manner. For example, this type of interface has been used as a communication medium that projects robot intent and task instruction to humans [12] so as to facilitate close collaboration between human workers and robot collaborators. However, in order to accurately project onto objects in the environment, this work assumed the 3D models of the objects were provided; unfortunately, these models are usually not available to end-users. In contrast to projecting robot intent, Ishii et al. used a projection interface for robot control with a laser pointer, and illustrated an iterative process for designing natural laser gestures [19]. Another relevant prior work is a point-and-click interface for specifying objects in the environment for robot manipulation [22], although it did not have a projection interface for the communication of robot and system states. Different from these prior works, this present one will use a projection-based interface for communication, action sensing, and task control.

3 PATI

In this section, we first present a "language" based on simple illustrations and gestures for robot programming. We then describe the PATI system, a proof-of-concept system that implements this illustration-based language to allow users to program a robot manipulator without substantial prior technical training or a background in robotics and programming.

3.1 Illustration-Based Language for Robot Programming

We envision an illustration-based language for task-level robot programming. We argue that this type of programming paradigm can increase the accessibility of robot programming and address a key challenge in programming robots to operate in flexible environments, in which objects and environmental landmarks are not predefined (as opposed to automation scenarios with predefined procedures and task configurations). Below, we describe an illustration-based language for robot programming, detailed in terms of syntax, semantics, and a composite program.

Syntax. At the syntax level, an illustration-based language supports various basic elements, such as tools and attributes. These elements can be specified and manipulated by users' gestural input. The examples described below are not exhaustive and can be expanded and modified to meet users' needs.

- Selection Tools: The Selection Tools can be used to select objects and define areas or landmarks in the workspace. These tools include common shapes (e.g., circles and squares) and free-form drawing (e.g., using a finger as a pen to draw a shape for selection). A Selection Tool has different attributes to denote different types of selections. For example, in PATI, an Object Selection Tool is represented as a circle, while an Area Selection Tool is in the shape of a square. Users can manipulate the Selection Tools using common gestures, such as a drag gesture to move them and a pinch gesture to resize and rotate the tools (Figure 2).
- Action Tools: An Action Tool specifies a task-level action that a robot can perform, and that can be attached to Selection Tools. For example, to move a group of objects from location A to location B, a user can select the objects using a Selection Tool, define a target area, and attach an Action Tool of "move-to" to the Selection Tool. A user may also need to specify other action-relevant information for the Action Tool; In the move-to example, the user should "link" the Action Tool to the target area. All of these specifications can be indicated through gestural commands.
- Attributes: Tools, objects, and the environment can be annotated with attributes; such attributes can be user-defined or system-generated. Attributes define the properties and behaviors of tools. As an example, a user can apply a color attribute to a "move-to" Action Tool to specify how objects in the Selection Tool may be sorted by their color (Figure 9 (a)).

Semantics. While syntax provides the basic tools and elements of the system, semantics describes their meanings and uses in a program. We note that the semantics of an element are customizable; for instance, an arrow can be defined to highlight a specific object or area of interest rather than only denoting the meaning of "move-to." The default semantics of an element can also be overwritten by an attribute applied to that element.

Program. A program is a composite of tools and their associated meanings. Users can use gestures to manipulate tools, annotate task-relevant landmarks in the environment, and compose programs. A program runs until its specified tasks are finished or it is terminated by the user. In the example of moving a group of objects from

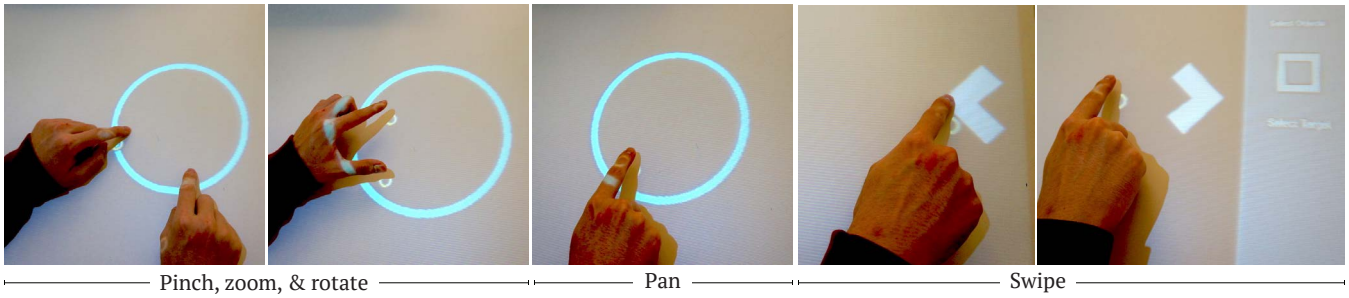


Figure 2: Users can use pinch, zoom, or rotate gestures to resize or rotate a tool, a panning gesture to move the tool, and a swipe gesture to open and close the Tools Menu.

location A to location B, the program executes multiple pick-and-place tasks until all the objects at location A are moved into the target area. We note that in this example, "looping" was implicitly embedded in the program specified by the user.

3.2 System Overview

The PATI system implements the illustration-based language described above, and allows users to program a robot manipulator directly on a table-top surface through an intuitive tangible interface. The system involves the use of a UR5 robot manipulator, which is a popular robot platform with a built-in programmable interface, a top-down projector, and a Kinect2 RGB-D camera mounted on the ceiling. The Kinect2 camera and the projector are calibrated to each other, but not to the table surface. Besides the hardware setup, the PATI system consists of four software modules: *Visual Perception*, *Tangible User Interface (TUI)*, *Program Synthesis*, and *Robot Control*, as illustrated in Figure 3. The modules of Visual Perception, Program Synthesis, and Robot Control were implemented in the Robot Operating System (ROS) [29]. The TUI was implemented in Unity.

The Visual Perception module detects and tracks objects in the environment, as well as a user's touch input, through the use of depth data and RGB color images from the Kinect2 camera. The user's input is then sent to the TUI module, in which different types of gestures are recognized (Figure 2). Once the user completes his/her program specification via the interface, the Program Synthesis module translates the user's task-level specifications into a set of robot commands, including gripper actions and the waypoint specifications of intended trajectories. The commands are then forwarded to the Robot Control module, which subsequently generates motion plans based on the input commands for robot execution. Below, we provide implementation details of the software modules.

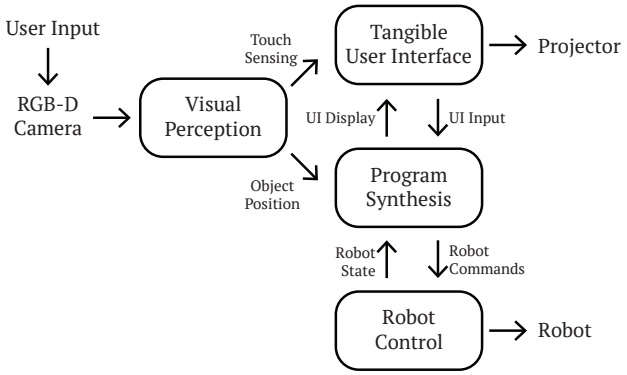


Figure 3: The PATI system is composed of four modules: *Visual Perception, Tangible User Interface, Program Synthesis,* and *Robot Control.*

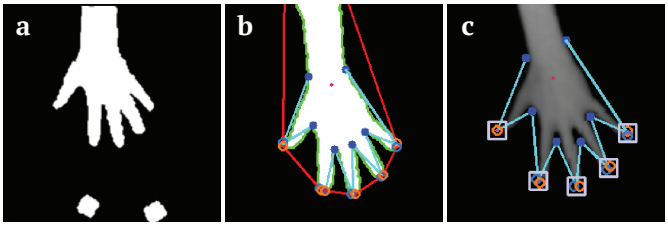


Figure 4: Fingertip detection. (a) Binary masks of hand and objects. (b) Contour and convex hulls are shown in green and red, respectively. (c) Detected fingertips based on the fingertip angle. The fingertip depths are the mean depth inside a surrounding window.

3.3 Visual Perception

The Visual Perception module handles the detection and tracking of objects and the user's gestural input.

Object Detection. We used the Point Cloud Library (PCL) [30] to detect objects on the table's surface. The input point cloud was first downsampled using a voxel grid filter. The resulting point cloud was further processed using the Random Sample Consensus (RANSAC) method [11] to identify the dominant plane surface in the point cloud; the dominant surface in our context was the table surface. We then performed Euclidean clustering on the cloud points that were above the table surface to identify and segment objects. The pose, shape, and size of the objects were recorded and used for object tracking. This process of object detection ran at approximately 10 frames per second, which was adequate in our context. The grasp position of each object was inferred from the shape of the object by finding the minor axis of the minimum enclosing ellipse.

Touch Detection. We used depth images from the Kinect2 to detect fingertips, following a processing pipeline similar to that

suggested by Xu et al. [39]. First, we generated a binary mask for everything detected above the table surface, including hands and objects (Figure 4 (a)), by thresholding the depth image. We then found the contour of each individual image blob, as shown in green in Figure 4 (b). To extract the hand, the blobs were further filtered using heuristics based on size, shape, and position on the table. Once the hand was extracted, we computed its convex hull and convexity defects. The convex hull, as depicted in red in Figure 4 (b), was defined as the smallest convex polygon enveloping the contour. As shown in Figure 4 (b), the fingertips were among the parts of the convex hull that overlapped with the contour.

To find the fingertips, we iterated through all the convexity defects (Algorithm 1). A convexity defect is a deviation in a contour from its hull and is represented by four parameters: the start point and end point of the hull edge from which the blob contour is deviating, the farthest point of the contour from the hull edge, and its approximate distance to the hull edge. In Figure 4 (b) & (c), start points are orange circles, end points are light blue circles, and the farthest points are the blue dots in between fingers. To detect fingertips, we calculated the angle between the farthest points of two neighboring convex defects. We empirically determined that 15 to 60 degrees of an angle represented a fingertip (Figure 4 (c)).

We used Algorithm 2 to determine whether or not users were interacting with the interface through gestural input. In brief, if the depth of a detected fingertip was under 15mm above the table surface, the user was using the interface. The depth of a fingertip was calculated based on the mean of a window around the fingertip, as illustrated in Figure 4 (c).

For effective tracking, the fingertips were registered between frames based on the difference in positions. To further obtain stable, smooth tracking, registered fingertips were filtered using the "1€ filter." This filter is a first-order, low-pass filter with an adaptive cutoff frequency [7]; it has a low cutoff frequency at low speeds, which reduces jitter, and a high cutoff frequency at high speeds, which reduces lag. The processed fingertips were passed to Unity continuously through the TUIO protocol designed for table-top tangible user interfaces [21].

Algorithm 1 Find fingertips from binary blob image

Require: cd: list of convexity defects
fts: an empty list of detected fingertips
function FindCandidateFingertips(cd)
for all $defect_i, defect_{i+1} \in cd$ **do**
$start_i, end_i, far_i, dist_i = defect_i$
$start_{i+1}, end_{i+1}, far_{i+1}, dist_{i+1} = defect_{i+1}$
$fingertip\_point = (end_i + start_{i+1})/2$
$fingertip\_width = Distance(end_i, start_{i+1})$
$left\_far = fingertip\_point - far_i$
$right\_far = fingertip\_point - far_{i+1}$
$fingertip\_length = Max(left\_far, right\_far)$
$fingertip\_angle = \cos^{-1} \frac{left\_far \cdot right\_far}{|left\_far||right\_far|}$
if $\begin{pmatrix} 15^\circ < fingertip\_angle < 60^\circ \ \& \\ fingertip\_width < 20mm \ \& \\ fingertip\_length > 30mm \end{pmatrix}$ **then**
Append $fingertip\_point$ to fts
end if
return $detected\_fingertips$
end for
end function

Algorithm 2 Find pressed fingertips

Require: img: the depth image
fts: list of the detected fingertips
d: window size
function FindPressedFingertips(img, fts, d)
$pressed\_fingertips = []$
for all $f \in fts$ **do**
$fingertip\_window \leftarrow$ window of size $d \times d$ at (f_x, f_y) in img
$fingertip\_depth = Mean(fingertip\_window)$
if $fingertip\_depth - tabletop\_depth < 15mm$ **then**
Append f to $pressed\_fingertips$
end if
end for
return $pressed\_fingertips$
end function

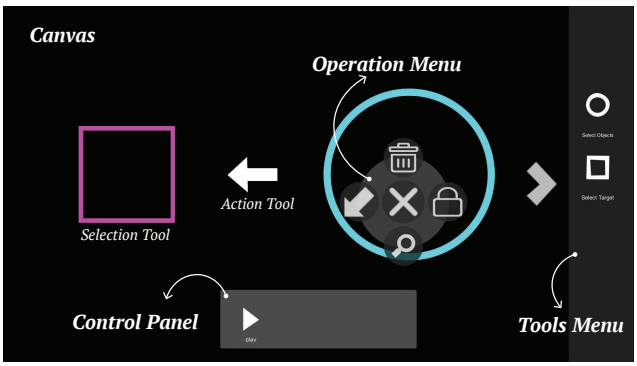


Figure 5: PATI's tangible user interface contains a *Tools Menu* for spawning tools, a *Control Panel* for program-related operations, and *Operation Menu* triggered by double tapping a Selection Tool.

3.4 Tangible User Interface

Our TUI module handled the projection of the user interface (Figure 5) and converted fingertip input from the perception module into common gestural commands. We utilized TouchScript[2], a multi-touch library for Unity, to implement multi-touch gesture recognition in PATI. Our current implementation supports common gestures for touch-based interaction, including pinch, zoom, rotate, pan, and swipe.

In our current implementation, the TUI has three main areas—the Canvas, the Tools Menu, and the Control Panel. The *Canvas* is the workspace where users can provide task specifications. The *Tools Menu* houses different tools that users can choose. In our current system, we use circles to represent Selection Tools for selecting objects of interest, and squares to represent Selection Tools for

[2]http://touchscript.github.io

specifying an areal landmark (e.g., a target area). Each Selection Tool has its associated Operation Menu, through which a user can confirm the selection, set attributes, and apply an intended action to the selection. The Operation Menu is triggered by double tapping the Selection Tool. Once the Tools are set and confirmed, the interface sends relevant Tool information (e.g., the position and shape of a selected object) to the Program Synthesis module using ROS#[3], an open-source library for communicating between ROS and Unity. Finally, after all the task-relevant specifications are made, users can launch the program using the "play" button on the *Control Panel*.

3.5 Program Synthesis & Robot Control

The Program Synthesis module generates high-level robot commands, whereas the Robot Control module creates low-level executable motion plans. When receiving information about confirmed user specification, such as object selection, the Program Synthesis module identifies and tracks objects inside the selection area by iterating through all the detected objects on the table and keeping track of the objects inside the selected area. When the user executes the program, the module generates a set of Cartesian waypoints of the end-effector's action trajectory.

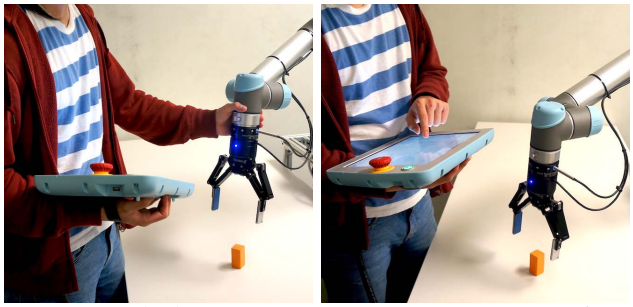
The generated waypoints for the pick-and-place of a single object typically include six end-effector positions: the pre-grasp position, the grasp position, the post-grasp position, the pre-place position, the place position, and the post-place position. The robot trajectory starts with moving from its current position to the pre-grasp position. The pre-grasp position is close to the desired grasp point, which is inferred from the pose, shape, and height of the target object. From the pre-grasp position, the robot adjusts the orientation of its gripper, if necessary, to reach the desired grasp point. After grasping the object, the robot then moves to the post-grasp position to avoid potential collision between the object and the table. Next, the robot follows a shortest non-collision path to the pre-place position. Once in the pre-place position, the robot orients the object to conform with the desired orientation, and places the object at its desired location on the table. It then moves to the post-place position and continues the rest of the task.

The Cartesian trajectory of the robot's movement (output from the Program Synthesis module) is sent to the Robot Control module. Joint angles of the robot manipulator are calculated by inverse kinematics on the Cartesian poses. The motion planning of in-between poses is handled by MoveIt![4]

3.6 An Illustrative Example: Pick and Place

Here, we use the pick-and-place task to illustrate how a user can utilize PATI to specify a table-top manipulation task. Figure 1 depicts this example in action. The user first opens the Tools Menu by swiping left from the right edge of the table. He then spawns a Selection Tool (a blue circle) by tapping the corresponding icon on the menu. He can further resize and rotate the Selection Tool using two-finger pinch gestures, and drags the tool to where the objects are. Following a similar procedure, the user can use a Selection Tool (a red square) to designate the target area. In this example of pick-and-place, the user then needs to specify an Action Tool (i.e., "move-to"), shown as a white arrow, that links the object selection and the target area. Upon the completion of the specifications, the user executes the program via the Control Panel. Other possible applications of PATI are shown in Figure 9 and further discussed in Section 6.1.

[3]https://github.com/siemens/ros-sharp
[4]http://moveit.ros.org



Programming by demonstration (e.g., physically moving the robot) | Programming using a UR5 teach pendant

Figure 6: A state-of-the-art method for robot programming in industrial contexts. The user programs the robot with a hybrid system of kinesthetic demonstration and a visual programming interface.

4 EVALUATION

In this section, we present a user study to assess the effectiveness of PATI in terms of task performance and usability. In particular, we compared PATI with a common programming by demonstration (PbD) method, kinesthetic teaching, provided by the manufacturer in its teach pendant (Figure 6).

4.1 Hypothesis

Our central hypothesis is that the PATI system will help participants achieve greater *task performance* in simple manipulation tasks such as pick-and-place, and offer higher *usability* to participants than the built-in PbD method for the UR5. Specifically, we pose the following two hypotheses:

Hypothesis 1: Participants will be able to program a UR5 robot manipulator to complete a pick-and-place task in a shorter time, make fewer mistakes, and ask fewer questions during their programming task when using the PATI system than when using the UR5 built-in PbD system.

Hypothesis 2: Participants will need a shorter practice time before they feel comfortable with the programming interface and report lower task loads when using the PATI system than when using the UR5 built-in PbD system.

4.2 Study Design & Experimental Conditions

We manipulated the interface that participants used to program a UR5 manipulator to complete a pick-and-place task. We designed a within-participants study with two experimental conditions as described below.

UR5 teach pendant: In this condition, participants used the PolyScope programming interface provided by Universal Robots[5] (the manufacturer of UR5) to complete the pick-and-place programming task. Based on the original 173-page UR5 manual, we created an abridged version of the manual and step-by-step instructions for the participants. In addition, a programming template was provided to the participants to show how to control the grippers and set waypoints for the robot's motion trajectory. This condition represents a state-of-the-art method for robot programming in industrial contexts (Figure 6).

PATI: In this condition, participants used the PATI system to complete the pick-and-place programming task (Figure 1). Similar to the UR5 condition, we provided participants with instructions on how to use the interface.

For both conditions, the experimenter would show the participants a demonstration video on how to use the respective interface in addition to providing its paper-based instructions. The order of the conditions presented to the participants was counterbalanced.

4.3 Practice & Experimental Tasks

Our evaluation focused on simple manipulation tasks (pick-and-place) fundamental to a variety of common functions that robots are envisioned to assist people in performing. Participants were allowed to practice using a given interface to program the robot to pick an object on a table and move it to a target area. The actual experimental task was similar to the practice task, except that it involved two objects as opposed to a single object.

Common PbD methods require users to physically move the robot manipulator to go through the waypoints of a desirable trajectory and to position the robot's end-effector appropriately so that it can successfully grasp an intended object. An example of such a programming paradigm is illustrated in Figure 6.

4.4 Measures

We used the following objective and subjective measures to assess task performance and usability of the PATI and UR5 interfaces for robot programming.

4.4.1 Task Performance. We developed three objective measures to assess the task performance of a robot programming interface.

Task Time: We defined task time as the time needed by a participant to complete the experimental task, starting from programming the robot task and ending when the robot finished the task. We note that the robot moved with the same speed in both conditions.

Number of Task Mistakes: We counted the number of mistakes participants made during the experimental task. A mistake was any unwanted effect produced by the robot, including the following situations:

- The robot grasped the object in an unstable configuration. This situation usually happened when a participant drove the robot's gripper too close to the table, presumably because they lacked a clear understanding of low-level robot operations (Figure 8 (a)).
- The robot failed to pick or place an intended object due to an inappropriate position or height of a waypoint (Figure 8 (b) & (c)).
- The robot—especially its gripper—collided with the environment, such as the table, or the other object during its task execution (Figure 8 (d)).

When a mistake led to subsequent mistakes, only the first was counted in our analysis.

Number of Questions Asked: We also counted the number of questions participants asked during the experimental task. We note that the experimental task took place after the participants felt comfortable with and had practiced using the interface for a pick-and-place task involving a single object.

4.4.2 Usability. In this evaluation, we focused on two aspects of usability: how much time users needed to be familiar with and comfortable using a new interface, and users' task load when using an interface for a task of interest.

Practice Time: We defined practice time as the time from when the participant made his/her first operation after the experimenter presented the practice task to when they indicated that they were ready for the experimental task.

Task Load: To measure task load, we adapted the NASA TLX [15] for our context. Specifically, we used three items—mental demand, physical demand, and effort—from the TLX scale to assess task load (Cronbach's $\alpha = .80$).

In addition to the measures of task performance and usability described above, we logged the *number of waypoints* participants recorded when using the UR5 interface. This measure serves as a proxy for understanding the participants' conceptual models of how the robot operates.

4.5 Study Procedure

After obtaining a participant's consent, the experimenter provided an overview of the study. The participant was randomly assigned to one of the two experimental conditions and given the necessary instructions and tutorial materials regarding the first programming interface. After viewing the materials, the participant was allowed to practice using the given interface with a simple pick-and-place task. During the practice session, the participant could review the instructional materials as they wished and ask the experimenter any questions they might have. The participant then moved on to perform the experimental task once they felt ready to do so. After the experimental task, the participant was asked to fill out a questionnaire regarding their experience using the interface, and was then introduced to the other programming interface. Similar to the procedure of using the first interface, the participant viewed relevant instructional materials, practiced using the interface, performed the experimental task, and filled out a post-interaction questionnaire. The study concluded with the experimenter collecting the participant's demographic information and interviewing the participant for additional comments. The study took approximately one hour and the participants were compensated with $10.

4.6 Participants

We recruited 17 participants (9 females and 8 males) on a college campus for this study. The participants were aged 23.48 years on

[5] https://www.universal-robots.com

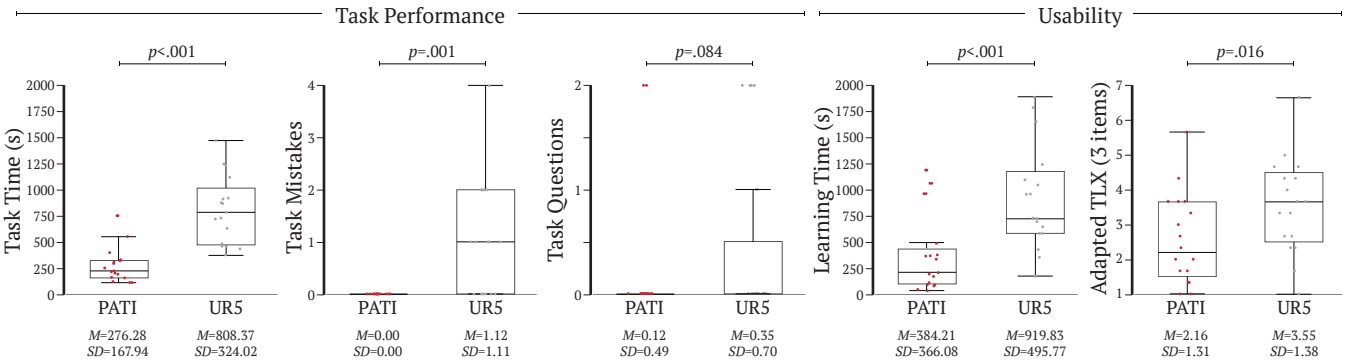


Figure 7: Box and whisker plots of data on the objective measures of task performance and the subjective measure of usability. The top and bottom of each box represent the first and third quartiles, and the line inside each box is the statistical median of the data. The length of the box is defined as the interquartile range (IQR). The ends of the whiskers are the first quartile minus 1.5 IQR and the third quartile plus 1.5 IQR.

average ($SD = 4.81$). The participants reported having familiarity with robots ($M = 4.00$, $SD = 1.97$) and programming ($M = 4.18$, $SD = 1.88$) on 1-to-7 rating scales (7 being most familiar). The participants had a variety of educational backgrounds, including engineering, psychology, writing, and medicine.

5 RESULTS

In this section, we report the results of our evaluation study, summarized in Figure 7. We used non-parametric Wilcoxon signed-rank tests according to the distribution of the analyzed data. For all the statistical tests reported below, we used an α level of .05 for significance.

5.1 Task Performance

Our first hypothesis stated that participants would be able to program a UR5 manipulator to perform a simple pick-and-place task in a shorter time, make fewer mistakes, and ask fewer questions during their task when using PATI than they would when using the built-in programming interface for UR5. Overall, our results supported this hypothesis.

Task Time. Our data showed that participants needed significantly less time to complete the experimental task when using PATI than they did when using the UR5 interface, $Z = 3.62, p < .001$.

Number of Task Mistakes. Our data indicated that the participants did not make any mistakes when using the PATI system, whereas they made more than one mistake on average when using the UR5 interface, $Z = 3.23, p = .001$.

Number of Questions Asked. We found that the participants asked the experimenter fewer questions when using PATI than they did when using the UR5 interface during the task, $Z = 1.73, p = .084$. We note that the observed difference was only marginal.

5.2 Usability

Our second hypothesis stated that participants would need a shorter practice time before they felt comfortable with the programming interface and would have lower task loads when using PATI than they would when using the built-in programming interface for UR5. Our results supported this hypothesis.

Practice Time. Our data showed that the participants needed significantly less time before they became familiar with the programming interface and felt ready for the experimental task earlier when using PATI than they did when using the UR5 interface, $Z = 3.34, p < .001$.

Task Load. Self-report data suggested that the participants experienced less task load when using PATI than when they used the UR5 interface, $Z = 2.41, p = .016$.

5.3 Quality of Task Demonstration

In addition to task performance and usability, we also measured the number of waypoints that participants recorded when they demonstrated the pick-and-place task of two different objects via the UR5 interface. The number of waypoints used varied among the participants, ranging from 5 to 13 points ($M = 9.12, SD = 2.62$). This variance showed that the participants had varying understandings of the robot's mechanical operations and motion plans. For the experimental task, fewer than 10 waypoints tended to cause mistakes. We next present common mistakes observed in the participants' demonstrations of the pick-and-place task.

5.4 Common Mistakes in Task Demonstration

One of the challenges in traditional robot programming by demonstration (e.g., kinesthetic teaching) is object/landmark referencing. Our results indicated that the participants using the UR5 teach pendant interface were likely to make mistakes in their programs; our observation of their programming sessions revealed some common mistakes. The most common mistake was object collision, where the robot's movement during its transportation of an object caused a collision with another object in the environment (Figure 8 (d)). This mistake was due to the manual teaching of trajectory waypoints and a lack of understanding of the robot's motion plans between the demonstrated waypoints. Without additional perceptual augmentation, traditional kinesthetic teaching is error-prone and has

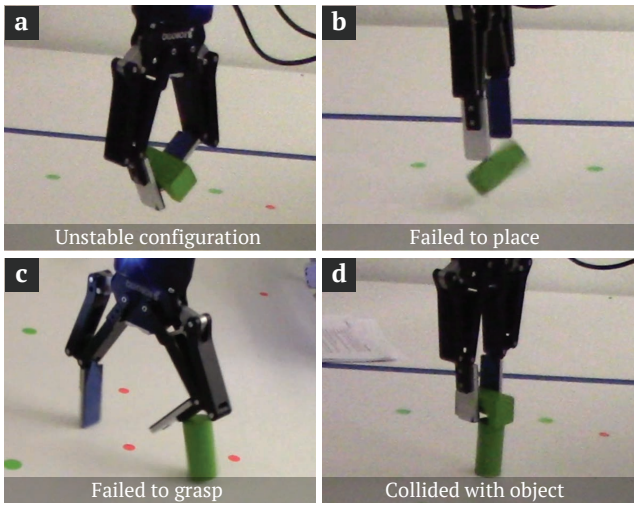


Figure 8: Examples of the three kinds of mistakes commonly made by the participants: (a) the gripper picks up the object in an unstable configuration; (b) and (c) the gripper fails to pick or place the object; (d) the gripper collides with objects while performing a task.

limited scalability. For example, a user would need to provide a kinesthetic demonstration for each object of interest.

Other common mistakes included moving the gripper too close to the table surface, thereby causing an unstable grasp of the object (Figure 8 (a)), and setting the release position way above the surface, thus causing the object to drop onto the table (Figure 8 (b)). These mistakes were all due to the lack of understanding of the operational mechanism of the manipulator and its gripper, which most end-users would not previously have. Some of our participants were master's students in a specialized robotics program. Even with technical training and prior experience of working with robot manipulators, they still made these common mistakes.

While some mistakes are less critical and only impact task performance, others may have serious negative effects, such as breaking objects or damaging the robot. For instance, one participant in our study accidentally drove the robot to hit the table. Therefore, it is crucially important to have an intuitive interface for end-user robot programming. The PATI system explores a promising interface for robot programming.

6 DISCUSSION

In this paper, we present an illustration-based method for robot programming and introduce PATI, which implements the method as a proof-of-concept system. The PATI system allows users to program a robot manipulator to perform simple table-top manipulation tasks directly on a work surface through the use of common gestures and tools. An empirical evaluation comparing PATI with a state-of-the-art method for robot programming by demonstration shows the effectiveness and potential of PATI for end-user robot programming. In particular, our results reveal that participants were able to learn how to use the programming interface in a shorter period of time, achieved a greater task efficiency, and had less task load when using PATI than they did when using the state-of-the-art PbD system. Below, we discuss possible applications that PATI can support, as well as the limitations and future directions of this work.

6.1 Applications of PATI

While this paper is focused on the presentation and discussion of PATI in the context of the pick-and-place task, the PATI system can easily be extended to support other various tasks and applications. Here we present four possible manufacturing scenarios that PATI can support: sorting, assembly, alignment, and inspection (Figure 9).

In a sorting scenario, users can use action attributes to specify how they would like a group of objects sorted. In the example in Figure 9 (a), objects are sorted based on the color of the action arrow. In the assembly example shown in Figure 9 (b), the robot sorts assembly parts into bins that the user can access. Once finished putting the parts together, the user places the assembled product into an outgoing area for the robot to fetch for the next step (e.g., inspection and kitting). Figure 9 (c) shows an example of object alignment, which underlies various kitting applications. Similar to the assembly scenario, the inspection example shown in Figure 9 (d) involves the robot bringing products to the user for inspection, as well as taking inspected products to the next processing step.

In addition to manufacturing applications, illustration-based robot programming may substantially reduce the technical barriers to utilizing personalized robot assistance for laypeople. For example, older adults may use simple gestures and tools to request assistance from their personal robots. Our future work involves deploying PATI in real-world settings to explore its possibilities and limitations.

6.2 Limitations & Future Work

While our results suggest the potential of illustration-based robot programming and the PATI system, the present work has limitations that highlight directions for future research. A key ingredient of touch-based interaction is the reliable sensing of user input.

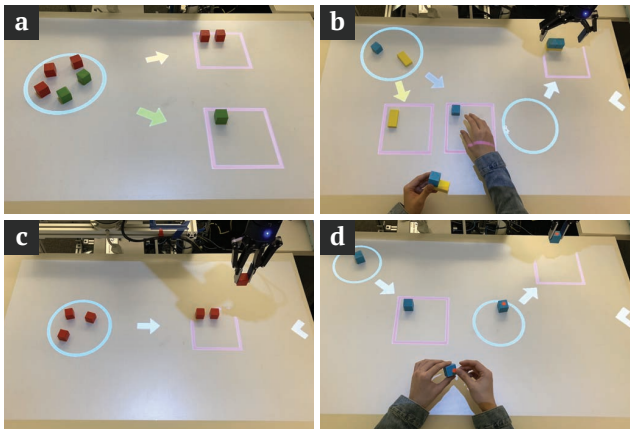


Figure 9: Possible applications of PATI include (a) sorting, (b) assembly, (c) alignment, and (d) inspection. The colored arrows in (a) and (b) represent how objects would be sorted based on color attributes.

Although the current implementation of PATI affords reasonable gestural interaction, it can be further improved. A participant commented, "*I really liked the new interface. [The PATI system] needs some tuning on finger recognition and gesture recognition, but [it] is clearly superior to the traditional UR5 interface for this task.*" To address this limitation, we will build on other successful methods for robust touch sensing (e.g., [38]) to ensure a quality user experience.

The possibilities of illustration-based robot programming are abundant. So far, PATI implements only a small set of gestures and task tools. Future work needs to explore both the breadth and depth of illustration-based robot programming by developing and evaluating new task tools, action operations, and input styles. Moreover, in addition to task-level specification, future work should explore how this novel programming paradigm may be used to craft motion-level and goal-driven social robot behaviors (e.g., [17]). Future research should also investigate ways to minimize environmental modifications (e.g., integrating the capabilities of projection and 3D sensing into the robot) and the needed system calibration to allow people to use it with ease in custom contexts. Finally, our evaluation involved a few participants who have some familiarity and experience with robotics. To truly understand the effectiveness of PATI for naïve end-users, future evaluations should involve users who do not have any technical background or prior experience.

7 CONCLUSION

In this paper, we present a Projection-based Augmented Tabletop Interface (PATI) for robot programming. PATI allows users to utilize common gestures and tools to specify robot tasks without coding. It also supports direct annotation in and reference to the environment to address the challenge of robot performance in a custom environment. A user evaluation showed that participants were able to learn and use PATI to program a robot manipulator to perform simple manipulation tasks efficiently and effectively. This work contributes to the broader vision of intuitive end-user robot programming.

8 ACKNOWLEDGMENTS

We thank Jaimie Patterson, Junlin Wu, and Ji Han for their help with this work.

REFERENCES

[1] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. 2012. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM, 391–398.

[2] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. 2015. Roboflow: A flow-based visual programming language for mobile manipulation tasks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 5537–5544.

[3] Rasmus S Andersen, Ole Madsen, Thomas B Moeslund, and Heni Ben Amor. 2016. Projecting robot intentions into human environments. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*. IEEE, 294–301.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 5 (2009), 469–483.

[5] Geoffrey Biggs and Bruce MacDonald. 2003. A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*. 1–3.

[6] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer, 1371–1394.

[7] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 2012. 1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems. *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems - CHI '12* (2012), 2527. https://doi.org/10.1145/2207676.2208639

[8] Sonia Chernova and Andrea L Thomaz. 2014. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8, 3 (2014), 1–121.

[9] Michael D Coovert, Tiffany Lee, Ivan Shindev, and Yu Sun. 2014. Spatial augmented reality as a method for a mobile robot to communicate intended movement. *Computers in Human Behavior* 34 (2014), 241–248.

[10] Rui Fang, Malcolm Doering, and Joyce Y Chai. 2015. Embodied Collaborative Referring Expression Generation in Situated Human-Robot Interaction. *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction* (2015), 271–278. https://doi.org/10.1145/2696454.2696467

[11] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395. https://doi.org/10.1145/358669.358692 arXiv:3629719

[12] Ramsundar Kalpagam Ganesan. 2017. *Mediating Human-Robot Collaboration through Mixed Reality Cues*. Ph.D. Dissertation. Arizona State University.

[13] Dylan F Glas, Takayuki Kanda, and Hiroshi Ishiguro. 2016. Human-robot interaction design using Interaction Composer eight years of lessons learned. In *Human-Robot Interaction (HRI), 2016 11th ACM/IEEE International Conference on*. IEEE, 303–310.

[14] Chris Harrison, Hrvoje Benko, and Andrew D Wilson. 2011. OmniTouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 441–450.

[15] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage Publications Sage CA: Los Angeles, CA, 904–908.

[16] Micha Hersch, Florent Guenter, Sylvain Calinon, and Aude Billard. 2008. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics* 24, 6 (2008), 1463–1467.

[17] Chien-Ming Huang and Bilge Mutlu. 2012. Robot behavior toolkit: generating effective social behaviors for robots. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*. IEEE, 25–32.

[18] Justin Huang and Maya Cakmak. 2017. 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*. ACM, 453–462.

[19] Kentaro Ishii, Shengdong Zhao, Masahiko Inami, Takeo Igarashi, and Michita Imai. 2009. Designing laser gesture interface for robot control. In *IFIP Conference on Human-Computer Interaction*. Springer, 479–492.

[20] Martin Kaltenbrunner and Ross Bencina. 2007. reacTIVision: a computer-vision framework for table-based tangible interaction. *Proceedings of the 1st international conference on Tangible and embedded interaction* (2007), 69–74. https://doi.org/10.1145/1226969.1226983

[21] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, Enrico Costanza, et al. 2005. TUIO: A protocol for table-top tangible user interfaces. In *Proc. of the The 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*. 1–5.

[22] Charles C Kemp, Cressel D Anderson, Hai Nguyen, Alexander J Trevor, and Zhe Xu. 2008. A point-and-click interface for the real world: laser designation of objects for mobile manipulation. In *Human-robot interaction (HRI), 2008 3rd ACM/IEEE international conference on*. IEEE, 241–248.

[23] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. 2010. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, 259–266.

[24] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, and Ewan Klein. 2002. Mobile robot programming using natural language. *Robotics and Autonomous Systems* 38, 3-4 (2002), 171–181.

[25] Tomas Lozano-Perez. 1983. Robot programming. *Proc. IEEE* 71, 7 (1983), 821–841.

[26] Cynthia Matuszek, Liefeng Bo, Luke Zettlemoyer, and Dieter Fox. 2014. Learning from Unscripted Deictic Gesture and Language for Human-Robot Interactions.. In *AAAI*. 2556–2563.

[27] Cynthia Matuszek, Liefeng Bo, Luke S Zettlemoyer, and Dieter Fox. 2014. Learning from Unscripted Deictic Gesture and Language for Human-Robot Interactions. *Proceedings of AAAI 2014* (2014), 2556–2563.

[28] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D Hager. 2017. CoSTAR: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 564–571.

[29] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.

[30] Radu Bogdan Rusu and S. Cousins. 2011. 3D is here: point cloud library. *IEEE International Conference on Robotics and Automation* (2011), 1 – 4. https://doi.org/10.1109/ICRA.2011.5980567 arXiv:1409.1556

[31] Kristin E Schaefer, Edward R Straub, Jessie YC Chen, Joe Putney, and AW Evans III. 2017. Communicating intent to develop shared situation awareness and engender trust in human-agent teams. *Cognitive Systems Research* 46 (2017), 26–39.

[32] Yasaman S Sefidgar, Prerna Agarwal, and Maya Cakmak. 2017. Situated tangible robot programming. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 473–482.

[33] Lanbo She, Yu Cheng, Joyce Y Chai, Yunyi Jia, Shaohua Yang, and Ning Xi. 2014. Teaching robots new actions through natural language instructions. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*. IEEE, 868–873.

[34] Lanbo She, Shaohua Yang, Yu Cheng, Yunyi Jia, Joyce Chai, and Ning Xi. 2014. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. 89–97.

[35] Tom Williams, Saurav Acharya, Stephanie Schreitter, and Matthias Scheutz. 2016. Situated open world reference resolution for human-robot dialogue. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 311–318.

[36] Andrew D Wilson and Hrvoje Benko. 2010. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. ACM, 273–282.

[37] Robert Xiao, Chris Harrison, and Scott E Hudson. 2013. WorldKit: rapid and easy creation of ad-hoc interactive applications on everyday surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 879–888.

[38] Robert Xiao, Scott Hudson, and Chris Harrison. 2016. DIRECT: Making Touch Tracking on Ordinary Surfaces Practical with Hybrid Depth-Infrared Sensing. *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces - ISS '16* (2016), 85–94. https://doi.org/10.1145/2992154.2992173

[39] Yanan Xu, Dong-Won Park, and GouChol Pok. 2017. Hand Gesture Recognition Based on Convex Defect Detection. *International Journal of Applied Engineering Research* 12, 18 (2017), 7075–7079.

[40] Hendrik Zender, Geert-Jan M Kruijff, and Ivana Kruijff-Korbayová. 2009. Situated Resolution and Generation of Spatial Referring Expressions for Robotic Assistants.. In *IJCAI*. 1604–1609.

[41] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and Pieter Abbeel. 2018. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *2018 IEEE International Conference on on Robotics and Automation (ICRA)* (2018).