Sketching Robot Programs On the Fly

David Porfirio  
Naval Research Laboratory  
Washington, DC, United States  
david.porfirio.ctr@nrl.navy.mil

Laura Stegner  
University of Wisconsin–Madison  
Madison, Wisconsin, United States  
steberger@wisc.edu

Maya Cakmak  
University of Washington  
Seattle, Washington, United States  
mccakmak@cs.washington.edu

Allison Sauppé  
University of Wisconsin–La Crosse  
La Crosse, Wisconsin, United States  
asauppe@uw.lax.edu

Aws Al barghouthi  
University of Wisconsin–Madison  
Madison, Wisconsin, United States  
aws@cs.wisc.edu

Bilge Mutlu  
University of Wisconsin–Madison  
Madison, Wisconsin, United States  
bilge@cs.wisc.edu

ABSTRACT
Service robots for personal use in the home and the workplace require end-user development solutions for swiftly scripting robot tasks as the need arises. Many existing solutions preserve ease, efficiency, and access through simple programming interfaces or by restricting task complexity. Others facilitate metacognitive task design but often do so at the expense of simplicity and efficiency. There is a need for robot programming solutions that reconcile the complexity of robotics with the on-the-fly goals of end-user development. In response to this need, we present a novel, multimodal, and on-the-fly development system, Tabula. Inspired by a formative design study with a prototype, Tabula leverages a combination of spoken language for specifying the core of a robot task and sketching for contextualizing the core. The result is that developers can script partial, sloppy versions of robot programs to be completed and refined by a program synthesizer. Lastly, we demonstrate our anticipated use cases of Tabula via a set of application scenarios.

CCS CONCEPTS
• Human-centered computing → Systems and tools for interaction design  
• Software and its engineering:

KEYWORDS
human–robot interaction, end-user development, sketching

ACM Reference Format:

1 INTRODUCTION
End-user development (EUD) solutions for robotics must allow end users to easily and efficiently create robot applications to satisfy immediate needs. Consider an example in which the manager of a grocery store must direct traffic away from a spill in the beverage aisle—a perfect task for a robot to perform and a seemingly simple task to specify. The manager must direct the robot to the location of the spill while ensuring that the robot avoids the spill; the robot must issue a cautionary statement to anyone approaching the aisle; and the robot must return to its charging station when the spill has been cleaned up (Figure 1).

Although simple in concept, designating the robot’s task at a moment’s notice may prove challenging. While learning techniques promise assistance due to their effectiveness in robotic task training, offline training forgoes critical social and environmental context, while online training takes time. End users, by contrast, already possess the contextualized knowledge required to specify a task. Therefore, we posit that programming tools for end users offer a better approach for on-the-fly task specification. Existing tools for robotics, however, are impractically instrumented for addressing immediate needs (e.g., desktop interfaces), or demand basic programming knowledge that is tangential to the expertise and skills of domain experts (e.g., automata or block-based programming in [21, 36]). Other programming paradigms, in contrast, compensate by restricting expressiveness, such as those that offer simple program representations (e.g., trigger-action programming in [26]), or those that limit developers to programming only one aspect of a robot’s behaviors (e.g., movement but not task goals in [53]).

To address the on-the-fly programming needs of end-user developers, we created a novel, on-the-fly, EUD solution, Tabula, designed to reconcile simplicity and expressiveness. The guiding principle of Tabula is to capture and automatically refine rapid, incomplete developer input from as minimally instrumented of an interface as...
possible. Achieving this goal is founded on two design choices. First, a formative design study conducted by the authors and described in this paper suggests that a multimodal interface with partial reliance on speech will enable end users to easily and efficiently express simple tasks for a robot to perform. Second, at present day, touch screen and voice interfaces such as mobile phones, tablets, and smart watches, are ubiquitous. End-user developers can therefore conveniently access touch interaction to contextualize spoken language statements and fill in logic gaps.

Guided by these ideas, Tabula enables end users to program robots through multimodal speech and sketching input. In a recording session, developers utter one or two spoken language statements that correspond to the primary goals, or core, of a task. To contextualize the core, developers sketch program logic on a two-dimensional representation of the robot’s target environment. When the recording session ends, Tabula’s program synthesizer leverages automated planning techniques to assemble a task by (1) embedding the robot’s goals within the path drawn by the developer and (2) inserting any additional steps required to achieve these goals. If multiple recordings have been provided, the synthesizer combines all the resulting task plans into an executable automaton.

Our primary contribution is therefore a programming system, Tabula, and the EUD paradigm that it affords. In this paper, we first describe a formative design study with a speech-only prototype, which ultimately served as a catalyst for the ideation of Tabula. We then describe Tabula itself, focusing on the integration of sketching to contextualize speech and specify program logic. Our contributions are summarized as follows:

- **System** — a full-fledged development tool, Tabula, and a set of application scenarios to demonstrate its use.
- **Design** — a design study that results in design principles for creating on-the-fly EUD tools for robots.
- **Technical** — a program synthesis approach for contextualizing spoken language with program sketches.

2 RELATED WORK

Our work draws on the literature from end-user development, natural language programming, program synthesis, and planning in artificial intelligence (AI planning). In the following, we briefly discuss relevant key concepts and related work.

2.1 End-User Development

End-user development (EUD) aims to democratize programming for novices. Lieberman et al. [27] characterizes EUD as surpassing application parameterization and customization and allowing users to modify or create programs from scratch. An EUD paradigm of note, trigger-action programming (TAP), has been widely successful in its adoption by end users [48]. However, despite its simplicity, TAP developers are still susceptible to inserting undesirable or unpredictable behaviors into their programs [56]. In a different approach to EUD, sloppy programming has explored the automatic mapping of coarse text entry to the capabilities of an API [28]. Under the umbrella of the no-code movement, a recently popularized term for EUD, many commercial products allow users to create complex applications, including Webflow for intricate webpages [50], AirTable for databases [1], and Zapier for automation [54].

Various approaches to EUD have been explored in robotics, but are typically limited in expressive power. These limitations have arisen from restrictive programming paradigms like TAP [26, 41] and input methods like natural language [15], or from making only a small subset of robot actions available to be programmed (e.g., only motion trajectories as in [53]). More expressive EUD interfaces, however, may increase developer mistakes and compromise the robot’s dependability. Prior work in end-user software engineering (EUSE) has sought to preserve dependability by providing end users with standard software engineering practices (e.g., fault localization) [7], and thus may prove useful for robotics.

Skeletal programming is a familiar concept in robot EUD and control. An especially natural use of sketching involves specifying the navigation path of a robot and the surrounding environment [5] or other navigation-related commands such as a drawn “X” indicating “go here” [43] or a drawn lasso indicating “vacuum this area” [40]. Tabula draws heavily from Roboshop, an interface for annotating a top-down view of a robot’s environment with tasks to perform [29], and V.Ra, a task-authoring interface that integrates navigation paths with both robot actions and program logic [8]. Additionally, similar to Tabula is the work of Shah [42] that integrates speech with sketching for specifying navigation commands and the work of Correa et al. [12] and Teller et al. [45] on a real-time robot control interface that also integrates speech with sketching. Among these works, Tabula derives novelty from its ability to synthesize branching and looping programs from coarse, on-the-fly multimodal input.

2.2 Natural Language Programming

Motivated by the widespread use of language in human interactions, researchers have explored several different approaches to allow natural language interactions with robots. Semantic parsing, in which natural language is transformed to a logical representation [13, 51, 55], has often been used to enable language specification of commands, goals, or simple programs [9, 32, 47, 49]. Alternative approaches based on syntactic features have also been shown to be effective when matched with appropriate domain knowledge [46]. In certain applications, the direct mapping of a controlled subset of English to the target formalism has proven sufficient [24].

Natural language dialogue systems use multi-turn language interactions to better accommodate the communication of complex instructions. In robotics, these systems have enabled end users to specify reusable programs for tasks such as navigation [25, 47], assembly [15, 44], and social interaction [18]. Some have envisioned human-robot dialogue as a way for future domestic robots to acquire necessary environment-specific knowledge, from the actions needed to complete some task to the rules that underlie the world [11]. In this vein, Mohan and Laird [34] developed an explanation-based task learning approach, using situated instructions to teach novel hierarchical tasks to a robot, and later work showed how these task representations could generalize across situations [23].

2.3 Planning & Synthesis

Program synthesis is used to automatically construct fully executable programs from partial developer specifications [19]. In human-robot interaction, program synthesis has been applied in both robot manipulation [16, 20] and social domains [10]. Similar
to Tabula, the programming tool Figaro synthesizes robot programs from multimodal speech and touch demonstrations [38]. Figaro, however, requires developers to recite their speech and touch in the exact order that they must occur in the resulting program.

To synthesize programs, Tabula uses techniques from AI planning, which is broadly defined by Alterovitz et al. [3] as “computing actions and motions for a robot to achieve a specified objective.” In accordance with Ghallab et al. [17], we classify our approach as operating at the descriptive level, in which plans contain information about what actions the robot performs and when to perform them, rather than the operational level that describes precisely how the robot should perform these actions. Tabula draws inspiration from notable successes of planning in human-robot interaction, including the Human-Aware Task Planner that plans a robot’s actions in accordance with social rules [2] and work from Petrick and Foster [35] that plans the actions of a social bartender robot for multi-party human-robot interactions.

3 SPEECH PROTOTYPE: ONE MODE OF INPUT

In this section, we describe our prototypical speech interface that ultimately served as a catalyst for the development of Tabula. In the vein of affording end users as much control with as minimal input as possible, the prototype explores the feasibility of end-user development with a single input modality—speech—for two reasons. First, speech is an intuitive form of communication inherent to everyday interaction. Second, sensing speech requires minimal instrumentation (i.e., only a microphone) beyond the robot itself.

In what follows, we describe the prototypical speech interface, its evaluation, and key lessons that inform Tabula.

3.1 Prototypical Speech Interface

The prototype consists of an early version of Tabula’s verbal input interface consisting of a wakeword recognizer, a speech-to-text engine, and a speech classifier. In the prototype, the verbal interface is accompanied by a simple visual feedback interface.

To use the verbal interface, end users begin designing a task by saying the wakeword “listen to me.” Then, users can verbally enter utterances into the interface, each of which is assigned to individual commands from an available set—either (1) action commands, which specify that the robot must do something, or (2) event commands, which specify that the robot should listen for a particular trigger to which the robot can respond, such as someone approaching or speaking. Instantiating event commands is thus the primary means for end users to encode human behavior in a program. For the prototype, we developed a small and exploratory set of commands. A sample subset of action (top five) and event (bottom two) commands is listed below:

\[
\begin{array}{ll}
\text{moveTo: } & \text{place } \rightarrow \text{ move to place} \\
\text{put: } & \text{item, place } \rightarrow \text{ put the specified item in place} \\
\text{say: } & \text{speech } \rightarrow \text{ say the contents of speech} \\
\text{ask: } & \text{speech } \rightarrow \text{ ask the contents of speech} \\
\text{tell: } & \text{narrative } \rightarrow \text{ recite the contents of narrative} \\
\text{eventApproach: } & \text{narrative } \rightarrow \text{ person approaches the robot} \\
\text{eventSpeech: } & \text{speech } \rightarrow \text{ person says speech to the robot}
\end{array}
\]

For the remainder of the paper, we refer to a command as a fully instantiated action or event in which all parameters in the command are resolved. A command type refers to an uninstantiated command. For example, the type of \text{say: } 'hello' \text{ is say, while the parameter of the instantiated command is 'hello.'}

To infer a command from an utterance, the prototype uses a non-learned, keyword-based approach that scores commands based on how well verbs and nouns in the utterance match a command’s type and parameters, respectively. Scores are derived by querying keywords—verbs, nouns, command types, and parameters—within WordNet [14, 33] and extracting the real-value distances between synonyms of these keywords. For example, within the utterance, “Put the groceries in the kitchen,” the action command \text{put: groceries, kitchen} scores highly because the words “put,” “groceries,” and “kitchen” match the command type and parameters.

Event commands score higher than action commands if the utterance contains keywords like “if” or “when.” For example, in the utterance “When someone says ‘hello,’” the event command \text{eventSpeech: 'Hello' (someone greets the robot) scores higher than its corresponding action command say: 'Hello' (the robot says ‘hello’) because the utterance begins with the word ‘when.’}

For speech commands, we require that the user provide the exact speech that the robot should utter or the exact speech that the robot can recognize. For example, if the end user wishes to specify that the robot emits a greeting, the user can say something like “The robot should now say ‘Hello, it’s nice to see you!’” in order to produce the corresponding action command \text{say: ‘Hello, it’s nice to see you!’}

As end-user developers produce a sequence of utterances, the prototype produces a program that consists of the corresponding sequence of commands. The sequence of commands is displayed on the visual feedback interface for the user to check. For editing in-progress command sequences, the prototype contains three simple directives: “undo” for undoing commands, “redo” for redoing commands, and “reset” for deleting all commands and starting over.

3.2 Formative Evaluation

To evaluate our design decisions within the prototype, we conducted a remote user study over separate video calls with five participants (three males, two females) aged 18 to 43 years (\(M = 24, SD = 10.7\)). Participants had little to no experience with robots and mixed levels of programming experience. The study was approved by an institutional review board (IRB).

In the study, participants were trained to use the prototype and presented with three tasks within a simulated home environment (e.g., welcoming someone home). For each of the three selected tasks, participants were allotted three minutes to program the robot and test the robot in a low-fidelity simulator within which participants could execute their programs over the video call. In the test environment, an icon of a robot moved around the home and interacted with participants via microphone and speaker.

At the end of the study, we asked participants to respond to the System Usability Scale SUS (10 items on a five-point rating scale) [6] and the USE questionnaire [30], which measures usefulness, ease of use, ease of learning, and satisfaction (30 items on a seven-point rating scale). The prototype’s average SUS score was 77 (\(SD = 17.9\)). Within USE, on a scale of one to seven, participants rated the

\[
\]
The design study with the prototype illuminates various challenges that end-user developers face with speech as their sole input modality. Due to the implications of the study, in addition to prior work that highlights various benefits of multimodal interfaces (e.g., inclusiveness and accessibility [52]), we supplemented speech with an additional modality, sketching, to create Tabula. We chose sketching in order to deemphasize speech by enabling end users to tactfully contextualize a small set of core, possibly underspecified commands. Sketching further allows end-user developers to craft program logic (e.g., loops) that are difficult to express verbally and maintains our goals of requiring minimal instrumentation for developers to complete a development task.

The Tabula system is implemented within two components communicating over ROS Noetic—a handheld touch or stylus-based interface implemented in Unity version 2020.3.21f13 and a synthesizer implemented in Python 3 (Figure 2, top). Given a two-dimensional map of the robot’s environment (Figure 2 bottom, a) with labelled regions (Figure 2 bottom, b), users verbalize a set of core commands and sketch the intended path of the robot on the map (Figure 2 bottom, c). Subsequently, the interface sends the recording consisting
of the user’s speech and sketch to the synthesizer, which returns a program to the interface.\(^4\)

In what follows, we describe (1) how Tabula is configured for use, (2) how users then create recordings from speech and sketches, and finally (3) how programs are synthesized from recordings.

### 4.1 Getting Ready to Use Tabula

Consider the following motivating example: a user wishes to program a robot to meet them every time they return from grocery shopping to help with unloading. In order to use Tabula, technical requirements must be satisfied, i.e., provide underlying assumptions of the robot’s capabilities and populate a map to use with Tabula.

**Robot Assumptions.** The developer must have access to a robot that is capable of creating a two-dimensional map of its environment, within which it should be able to accurately localize itself and recognize objects. In addition, the robot must be equipped with state-of-the-art path, motion, and task planners—it should be able to navigate to different areas in the environment, interact with objects that it recognizes, and handle edge cases in its task within reason (e.g., if the robot has a goal to grab groceries but the groceries are inside of the user’s car, the robot will know to open the car door and search for the groceries before grabbing them).

**Knowledge Handling.** Prior to use, Tabula must possess contextual knowledge. Knowledge handling within Tabula draws heavily from prior work in AI planning, particularly Petrick and Foster [35], in that Tabula contains a fixed domain that describes the universe of known possible entities that the robot is assumed to be able to recognize and interact with (e.g., types of objects and humans), the semantics of each entity (e.g., “cabinet” is a “container”), a set of available commands that consist of actions for the robot to perform or events that it should wait for, and preconditions that must be met to perform or post-conditions that hold true as a result of some commands. Also in accordance with common practice, Tabula stores current world state within a dynamic, modifiable world database.

**Map Setup.** Prior to specifying a task within the robot’s environment, end users may use Tabula to request the robot’s most up-to-date two-dimensional map (Figure 2 bottom, a). Then, Tabula is used to color regions of interest, or areas on the map that the robot is expected to visit (Figure 2 bottom, b). Finally, the user can use the interface to add objects to the map that may also be of interest to the robot. For instance, the user may place a “groceries” icon in the garage region, thus adding it to the world database and indicating to the robot that it can find groceries in the garage. The latter step of placing objects in regions is not a strict requirement.

### 4.2 Recording a Task

When an end user is ready to program their robot, they create a recording, shown in Figure 3a. A recording consists of one utterance \(\mu\) and one sketch \(\sigma\). The utterance is intended to describe the core of the task for the robot to perform, while the sketch is intended to ground the utterance within the robot’s surrounding environment.

Using our motivating example for illustration, when the end-user developer is ready to embark on their shopping trip, they pull out their phone, activate the Tabula app, and press the “Record” button. While recording, the end user’s first action is to verbalize the task core: “when I arrive, bring in the groceries.” Tabula uses the Stanford CoreNLP library [31] to detect “when I arrive” as a subordinate clause and splits the user’s speech accordingly into two separate parts. Then, Tabula parses each clause into individual commands, shown in Figure 3a-b, using a similar approach to §3.1 with a few notable differences. First, Tabula foregoes a scoring-based approach in favor of pure keyword matching to map nouns in \(\mu\) to command parameters and VerbNet [22] (rather than WordNet) to map verbs in \(\mu\) to synonyms of command types. Tabula also supports partially specified commands, such as commands that contain unfilled parameters. Given these modifications, Tabula parses “when I arrive” to a candidate event command \texttt{eventApproach} and “bring in the groceries” to a candidate action command \texttt{put: groceries \_\_\_\_\_\_}, in which the blank line represents an unspecified argument.

Occurring either before, during, or after verbalizing the task core, the end user sketches the sequence of regions that the robot should visit. Beginning in the living room region, the developer slides their finger to the garage region, then to the kitchen region, and then back to the garage. Tabula parses the sketch \(\sigma\) into the sequence of regions \(\text{garage} \rightarrow \text{kitchen} \rightarrow \text{garage}\), omitting the first location (living room) so as not to restrict the robot to begin its task in any one region on the map. Figure 3a-b depicts the step of parsing \(\sigma\) to a region sequence.

### 4.3 Program Synthesis and Output

Given one or multiple recordings provided by the end user, the goal of the synthesizer is to (1) create traces from each recording and (2) assemble a finite automaton, or program, that accepts each trace.

**Creating a Trace from a Recording.** Given a recording \(R\) containing a parsed utterance \(\mu\) and parsed sketch \(\sigma\), the synthesizer must create a trace \(t\) that satisfies the constraints set by parsing \(\mu\) and \(\sigma\). A trace is a sequence of robot actions \(a_0 \rightarrow a_1 \rightarrow a_2 \ldots \rightarrow a_n\) where \(a_i\) is the \(i\)th robot action and \(e_i\) is the \(i\)th event. Figure 3c illustrates the task of formulating a trace \(t\) from individual components \(\mu\) and \(\sigma\).

To illustrate, recall our example with the utterance “when I arrive, bring in the groceries” and the sketch from the living room to the
garage, to the kitchen, and then back to the garage. For clarity in describing how Tabula creates a trace for this recording, let us begin by considering a simpler example in which the garage is visited only once (we will return to our full motivating example in §4.3, Loops).

The utterance is still parsed to the commands eventApproach and put: groceries, ____. The utterance is still parsed to the commands eventApproach and put: groceries, ____

The utterance is still parsed to the commands eventApproach and put: groceries, ____. but the sketch is parsed to the shortened sequence of regions garage → kitchen. With our shortened sketch, the task of the synthesizer is to create trace t as follows, where unlabeled transitions refer to the empty event in which the robot needs no prompting to perform one action after another:

\[
\text{idle} \xrightarrow{\text{eventApproach}} \text{moveTo: garage} \xrightarrow{\text{grab: groceries}} \\
\text{moveTo: kitchen cabinets} \rightarrow \text{put: groceries, kitchen cabinets}
\]

In its search for trace t, the synthesizer must make multiple decisions autonomously: (1) within which regions the core commands from µ should be inserted, (2) how to resolve unfilled arguments from these core commands, (3) whether and where additional robot actions need to be inserted such that the preconditions of each command in the trace are satisfied, and (4) whether and how the world database needs to be modified such that the robot can complete the trace successfully. In order to make these decisions, the synthesizer employs A* search to plan for the most optimal trace in terms of discrete actions and locations. The planning space includes the following penalties:

1. Traces incur penalties equal to their length. Longer traces are thus more costly than shorter traces.
2. Each region or entity that the robot visits incurs an additional penalty if the robot does no action at that location.
3. Any entity that exists in the trace but has not yet been inserted in the world incurs an additional penalty.

The planning space includes the following additional constraints: the synthesizer will only accept traces that (1) include moveTo commands for each region present in the original sketch, and (2) include the core commands specified by the end user’s utterances. If an object exists in an accepted trace that does not yet exist within Tabula’s most up-to-date snapshot of the robot’s environment (the world database), the object will be added to the world database.

To illustrate the planning space within our shortened example, the synthesizer makes the following decisions. The eventApproach core command is inserted before the robot moves to the garage and the core put: groceries, ____ command is inserted when the robot is in the kitchen. In deciding how to resolve the put: groceries, ____ command with the unfilled argument for where the robot should place the groceries, the synthesizer searches for an entity in the domain labelled as “container” and existing in the kitchen region, and completes the command with the argument kitchen cabinets. In determining whether and where additional robot actions are needed in t, the synthesizer knows from the planning domain that a precondition of put is that the robot must first be holding an entity before it is able to put it somewhere. Therefore, the synthesizer decides to insert a grab: groceries command for when the robot is in the garage. Lastly, if the world database does not already indicate that groceries can be found in the garage, the synthesizer will modify world accordingly and incur a penalty.

Assembling a Program. Given a single trace, there may be nothing left for the synthesizer to do – the trace itself becomes a step-by-step program for the robot to execute. If the developer inserts loops into a recording or provides multiple recordings, then the synthesizer will have additional work.

Loops. Within a single recording, end users may introduce loops. To do this, end users need only visit a region multiple times in the course of a sketch. To illustrate, let us return to our original motivating example in which the recording still consists of µ “when I arrive, bring in the groceries” and σ once again consists of the robot moving from the living room to the garage, from the garage to the kitchen, and then back from the kitchen to the garage. As before, σ is parsed to the sequence of regions garage → kitchen → garage.

The synthesizer detects a loop within the sketch (garage is repeated) and then extends the loop such that there are two iterations total and each loop iteration is identical. Taking into account garage → kitchen as a single loop iteration, the sketch will be extended so that this iteration completes twice, producing the following modified sketch σ’: garage → kitchen → garage → kitchen.

For producing a trace from µ and σ’, an additional synthesis constraint is necessary—for any location (i.e., a region or entity) visited multiple times in a trace, the sequence of actions and events occurring at that location must always be the same. The resulting trace is therefore as follows:

\[
\text{idle} \xrightarrow{\text{eventApproach}} \text{moveTo: garage} \xrightarrow{\text{grab: groceries}} \\
\text{moveTo: kitchen cabinets} \rightarrow \text{put: groceries, kitchen cabinets} \\
\text{moveTo: garage} \rightarrow \text{grab: groceries} \\
\text{moveTo: kitchen cabinets} \rightarrow \text{put: groceries, kitchen cabinets}
\]

To assemble the final program, the synthesizer combines repeated sequences of actions and conditionals to form a loop, shown in Figure 3d.

Multiple Recordings. After an initial recording has been provided, additional recordings can be attached to any existing recording. Attached recordings cannot start from any arbitrary location in the world; rather, they must branch from a location within an existing sketch. The synthesizer assembles traces from each recording no differently than if only one recording was provided.

Assembling an executable program from an initial trace and one or more attached traces is straightforward. If the end user begins an attached recording at a location l with a core event command (i.e., “when I say ’stop helping me with the groceries’”), the resulting program will contain a branch at l in which the trace resulting from the attached recording will execute immediately when the event occurs. It is possible for nondeterminism to arise from the attachment of traces to each other, such as if the end user begins an attached recording without providing a core event command.

5 TABULA CAPABILITIES AND LIMITATIONS

We demonstrate Tabula’s capabilities by describing a set of application scenarios. Next, we utilize a suite of 33 total synthesizer test cases that cover, but are not limited to, different variations of these scenarios in order to provide an analysis of Tabula’s reliability.

5.1 Application Scenarios

In addition to the Grocery scenario introduced in §4, we demonstrate Tabula’s capabilities with three additional application scenarios.
Figure 4: Our application scenarios include alerting people to a spill (top), guiding people to the visitor center in a hospital (middle), and tidying up after a playdate (bottom). Drawn sketches are graphically enhanced for clarity.

Alerting People to a Spill. This scenario is identical to the one introduced in §1 where the manager of a grocery store needs to direct traffic away from a spill in the beverage aisle. Recall the task requirements that the robot must move to the location of the spill while avoiding the spill, issue a cautionary statement to anyone approaching the aisle, and return to its starting point after the spill is cleaned up. This application scenario demonstrates on-the-fly task contextualization and using Tabula’s branching and looping functionality to create trigger-action programs.

Figure 4 (top) depicts the steps taken by the manager to program the robot. First, the manager contextualizes the task by drawing a new region to indicate where the spill occurred. Next, the manager creates a recording to direct the robot from its starting point to the beverages, purposefully circumventing the spill so that the robot avoids driving through it (Figure 4a). To ensure that anyone who enters the robot’s vicinity is alerted to the spill, the manager then sketches a self-loop in the beverages aisle and utters “When someone approaches the aisle, say, ‘Please avoid the spill in this area. It will be cleaned shortly’” (Figure 4b). The effect of this recording is to create a trigger-action program that remains in effect while the robot is in the beverages aisle—whenever someone gets near the robot, the robot will alert them to the spill. Finally, the manager directs the robot back to its starting point by sketching a trajectory from the beverages aisle back to the cashiers and uttering “When I say ‘go home’” (Figure 4c). Figure 4d presents a decontextualized, high-level illustration of the resulting program.

Guiding Visitors in a Hospital. Consider an employee at a busy hospital wing who wants to streamline the check-in process for visitors. The robot should offer to escort visitors from the hospital entrance to the visitor center. This application scenario is intended to highlight how human behavior can be encoded into a program.

Figure 4 (middle) depicts the steps taken by the employee. First, the employee inserts a person entity in the entrance to the hospital, indicating to the robot that it will encounter people in this area. The employee then sketches a path from the entrance to the visitor center and utters, “Tell people the directions to the visitor center. Say, ‘would you like me to escort you there?’ ” (Figure 4e). Next, the employee utters, “If they say ‘yes,’ ” and in response to the robot hearing “yes,” sketches a path from the entrance to the visitor center and back to the entrance (Figure 4f). The resulting program, depicted in a decontextualized and high-level form in Figure 4g, will thereby loop forever in which the robot approaches people in the hospital entrance, asks them if they are interested in being escorted, and if so, escorts them to the desk.

Tidying Up. Consider a parent with toys scattered around their home after a playdate. The parent wants to program the robot to remove toys from three specific rooms in their home and place the toys in a chest in the living room. This application scenario is intended to demonstrate looping tasks and the synthesizer’s ability to place new objects in a scene.

Figure 4 (bottom) depicts the steps taken by the parent. The parent begins by uttering, “Put the toys in the chest,” and sketching a loop from the robot’s starting point to the bedroom, the living room, and then back to the bedroom. Based on this input, Tabula inserts a toy object into the bedroom and a toy chest object into the living room (Figure 4h). The user then provides the same utterance and sketches a path from the chest, to the kitchen, and back to the chest (Figure 4i). Finally, the parent provides the same utterance a last time while directing the robot to the hallway (Figure 4j). In the resulting program, depicted in a decontextualized, high-level form in Figure 4k, the robot loops on picking up toys from the bedroom, hallway, and kitchen until no toys remain.

5.2 Synthesizer Reliability

Our suite of 33 synthesizer test cases (referred to as T1-33) allows for a high-level analysis of Tabula’s reliability. Each test case provides the following input to the synthesizer: (1) the end-user’s speech, (2) the end user’s sketch, and (3) a custom world database tailored to the test case. Given this input, the synthesizer produces a program as output, taking an average of 3.04 (SD = 0.70) seconds per test case on an Intel Core i7-1065G7 CPU (1.30 GHz). Based on our experiences from constructing our test suite, we have observed three categories of reasons for which the synthesizer might fail to produce the intended output, which we detail below.

Insufficient Information from Speech. While Tabula is robust to omissions from end-user speech, failure to synthesize the intended program may occur if supporting information is also missing from the world database. Consider the Groceries scenario presented in §4.1, encapsulated in test case T8. Had the end user been vague in their speech (e.g., “bring them in,” rather than “bring in the groceries”), T8 will still produce the correct output if the groceries entity is present in the world database. However, if the
groceries entity is missing both from user speech and the world database, the synthesizer will nondeterministically choose an item from the domain to insert into the world for the robot to grab, which may not be groceries.

**Insufficient Information from Sketching.** Although much information about the robot’s core task is provided through speech, sketching provides contextualization of the task within the robot’s environment and information about program structure (i.e., branching and looping). Consider the Hospital scenario presented in §5 (T16). The scenario contains a loop in which the robot proceeds back to the entrance after escorting a visitor to check in. If the developer does not explicitly sketch the path back to the entrance, this loop will not be inserted in the program.

**Insufficient Domain Knowledge.** This category of failure pertains to the synthesizer not possessing enough prior domain knowledge for contextualization. For example, consider the Tidying application scenario presented in §5 (T26). Instead of cycling between picking up a toy and dropping it in the chest, perhaps the end user wants the robot to first collect all toys, and then drop them into the chest at once. The end user may correctly sketch a path through the kitchen, bedroom, and living room and say “pick up the toys.” In this case, however, the synthesizer does not possess enough prior knowledge about the different ways in which tidying can be performed and without this information directs the robot to pick up toys from a single room rather than each room.

6 DISCUSSION

6.1 On-the-Fly End-User Robot Development

There is a need for tools that enable end-user developers to rapidly and conveniently script robot programs for situations that arise spontaneously. Although robots are well-suited to handle these situations, development solutions that afford meticulously crafting highly contextualized applications may be difficult and slow to use. In contrast, hands-off techniques stemming from machine learning are highly effective at generating and refining robot applications, but the offline application of these techniques results in decontextualized task specifications, while the online application of these techniques in the intended interaction context requires arduous data collection. With Tabula, we posit that the best way to rapidly obtain contextualized information about a task at hand is through simple forms of input from end users themselves, who represent domain experts within the robot’s target context.

We believe that Tabula represents a significant step in this direction. First, Tabula is quick to use. As demonstrated by our application scenarios and test cases, a full application can be developed using, at minimum, a single speech utterance paired with a single sketch. Furthering its versatility, Tabula requires very little instrumentation other than a robot and a personal mobile device. Furthermore, Tabula enables task contextualization, owing to the ability to customize the robot’s environment and ground spoken language commands within this environment. Users can therefore apply Tabula to create a robot program in any situation in which the robot is able to localize within its environment. Lastly, Tabula handles complexity without requiring users to pore over task details. With a few simple spoken language commands and the high-level program logic derived from the user’s sketch, Tabula synthesizes a finite state automaton.

6.2 Limitations & Future Work

A key limitation of Tabula is its lack of evaluation with potential robot end users. As such, we cannot conclude whether Tabula is more effective than existing state-of-the-art solutions for scripting contextualized robot applications on the fly, and we cannot offer conclusive design implications for how Tabula may be improved. Plans for additional data collection are therefore underway.

Second, although Tabula is intended for non-programmers and technical non-experts, we still believe that end users must be trained on how to use Tabula to its full potential. In particular, we expect that forming a loop within a single recording or creating a branching program from multiple recordings will require practice. Furthermore, minimal training may be required for end users to learn how to optimally specify goals via natural language. Conducting a qualitative, exploratory evaluation of Tabula will enable us to understand precisely where training is required.

Third, Tabula lacks in offering feedback to end users and the ability to refine and correct programs, both of which are critical to successful human-AI systems [4]. Future work must first provide end users with information about potential faults or unexpected program behavior, such as branches with underspecified triggering events, and then provide end users with the means to correct these issues. Correcting issues will necessitate expanding Tabula’s refinement capabilities, such as by allowing end users to target and fine-tune specific aspects of a recording.

Fourth, as described in §4.1, Tabula requires domain knowledge, including a map of the environment, prior to task specification. While Tabula already somewhat challenges this requirement—environment mapping may be achieved during, rather than prior to, task specification if the end user sketches paths to unmapped areas—interacting with entities not already in the domain is not yet supported. Future work on Tabula should integrate auto-classification of novel entities within Tabula or modify Tabula’s user interface to prompt the end user to classify these entities for the robot.

7 CONCLUSION

We present Tabula, a system for on-the-fly end-user development of robot programs. Tabula is motivated by the need for simple programming interfaces that maintain the expressiveness required for robot development. We thereby approached the design of Tabula from the ground up, beginning with an initial speech-only prototype. Based on the results of a design study, we created Tabula to supplement speech with an additional mode of input, sketching. In a series of application scenarios, we demonstrate how Tabula can create meaningful robot programs through speech and sketching.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) award 1925043, NSF Graduate Research Fellowship Program award DGE-1747503, and a Cisco Wisconsin Distinguished Graduate Fellowship. This work was carried out while DP was affiliated with the University of Wisconsin–Madison and completed while an NRC Postdoctoral Research Associate at the Naval Research Laboratory.


