Rapid Prototyping of Human-Robot Interactions via Verbal Authoring

ABSTRACT

Robotic products are becoming increasingly commonplace. Introducing these products into day-to-day settings involves an interaction designer to "author" the robot's behavior. Traditionally, the designer authors interactions using connected visual nodes that model behaviors. Visual authoring enables designers with little programming experience to design robot behaviors but still requires the designer to learn the environment's conventions and the setup of the environment-robot system, which can become barriers to rapid prototyping and design. In this paper, we present an authoring approach that aims to remove these barriers and provides the designer with the ability to create and implement robot behaviors using natural spoken language. We implemented our verbal authoring tool and evaluated the effectiveness of and user experience with the tool by comparing it against a popular visual authoring system called Choregraphe. Based on data from 24 participants, our findings show our verbal-authoring approach to improve usability and task performance in measures of task-completion rate and task time, without increasing user task load compared to a state-of-the-art graphical approach.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces

Author Keywords

Authoring systems; visual programming; verbal authoring; speech-based interfaces; human-robot interaction; robot programming

INTRODUCTION

The field of human-robot interaction (HRI) has been significantly shaped by the creation and use of authoring tools in the field. Prior to this introduction, HRI research was limited to those with prior computer programming experience. Authoring tools have allowed non-technical experts to enter the HRI field [12], as their creative content could be integrated with a particular technical system. Traditionally, the design of authoring tools has been graphics-centered. Choregraphe, Aldebaran Robotics's robot behavior authoring tool for the humanoid robot Nao, is an example of a state-of-the-art graphical authoring tool [17]. The "drag and drop" gesture of different graphical blocks representing sensory input, actuation, and logic control, is a core feature of authoring tools like Choregraphe for programming a robot behavior. Due to the mainstream popularity of graphical authoring tools, the study of non-graphical tools is a subject of interest.

Figure 1. We propose a "verbal" protocol for authoring robot behaviors to enable rapid prototyping or programming of robotic products. An experimenter is shown authoring a supermarket interaction on a NAO robot (top). A user interaction with the authored robot is also shown (bottom).

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Currently, there is relatively limited research on the viability of speech-based authoring tools over graphical tools and their application to HRI. Nicolescu and Matarić [16] describe a verbal system as a more simple and engaging way of authoring robot behaviors. Using one's voice to say what a robot should do may be as simple as using a drag and drop gesture in a graphical environment. The authoring system developed by Nicolescu and Matarić [16] uses verbal cues as an aid to augment and reduce the complexity of a physical behavior a robot needed to learn via human demonstration. The researchers found that using verbal cues was a successful approach in helping the robot distinguish between relevant and irrelevant behaviors, concluding that speech formed an important feature in behavior-learning feedback [16].

With the significance of speech in behavior-authoring in mind, we decided to create a verbal authoring software to create, edit, and test robot behaviors. Verbal patterns for the software were inspired by Sauppé and Mutlu [18] on design patterns for HRI. Their paper laid out seven basic "design interaction patterns" that frequently comprised behavior-authoring dialogue (e.g. an "instruction-and-action" pattern) and helped shape the manner in which the author and end-user of our system communicate with the robot. The overall design goal of our system is to enable high-level behavior-building capabilities with speech recognition on a limited and pre-defined "verbal domain"—a set of natural phrases and phrase templates with corresponding code-level implementation. Completed behaviors are discussed within the context of verbal interaction which involve another "end user" human which is not the author.

A given behavior authoring task may involve mutable states, sensory input, and conditions. The complexity of a certain behavior increases with the required elements needed for a robot to complete the task. A study by Cohen, Horowitz, and Wolfe demonstrated that auditory memory is inferior to visual memory [6]. As authoring more complex tasks requires frequently recalling what had been previously appended to or changed in a behavior, authoring in a dialogue-based scheme may be more difficult than in a visual environment. We hypothesize that a verbal tool will allow authors to edit less complex interaction behaviors with a higher time efficiency and higher standard measures of user satisfaction and perceived workload than a state-of-the-art graphical authoring environment, Choregraphe, could. The steep learning curve of mainstream graphical authoring tools presents a needless overhead for simpler tasks which involve little use of hidden variables or states. In the rest of this paper, we explain our implementation of a verbal authoring tool that satisfies our research interest criteria. We then present our evaluation of the viability of our tool for authoring tasks using standardized measures of completion and usability. The large-scale goal of this project is to create a new way for lay-people, not necessarily technical experts, to author interaction behaviors for robots. We anticipate this goal to allow our system to be accessible across industry, education, and research.

RELATED WORK

In the following paragraphs, we look at related work on design patterns for authoring tools, embodied computing, real-time environments, and alternative authoring tools for robot behaviors.

Design Patterns

There are certain features, or design patterns, that should be considered when creating an authoring tool; the search for effective design patterns is well documented in various research papers. A common feature for authoring tools is to support dynamically mutable variables in a runtime environment using a graphical interface [14, 10]. Expert tools for computer programmers such as Juxtapose use a tuner to allow "changes to variables" [14]. Even novice tools such as the "interaction composer" for non-programmers on HCI research teams use "decision blocks" to manipulate runtime behavior [10]. d.tools provides a *statechart editor* that changes during prototyping interactions between sensor hardware and different outputs [13]. Intended use alike, dynamically mutable states are prevalent as a design pattern. Having a tunable runtime environment allows for lower-effort and quicker parameter experimentation [14] since changing parameter values during runtime rather than halting and changing source code is more convenient for rapid prototyping.

Embodied Computing

Embodied computing in HCI involves developing an authoring environment tailored to hardware devices and robotic systems. Research in this area gives insight on development and testing of hardware behavior, computational literacy in constructivist education, and presentation and notation of an authoring language for novice programmers.

d.tools and Choregraphe illustrate similar concepts for embodied computing. Research with d.tools focuses on how design teams can rapidly prototype information appliances, with an emphasis on prototypes that are iteratively "designed, tested, and analyzed" as author design hypotheses [13]. The d.tools environment supports development on three levels, the "hardware-to-PC interface, intra-hardware communication level, and the circuit level" [13]. Development by *doing* rather than by *thinking* is an underscored aspect of this research and has important connections to constructivist education in computational literacy. Choregraphe, is a graphical module of Aldebaran Robotics's NaoQi, a distributed environment to run and allow communication between multiple binaries, centered around development for the humanoid robot Nao. Choregraphe graphically represents NaoQi's distributed functions, displaying a flow chart of behaviors and a timeline that can be seen in the actions of the Nao associated with that instance of Choregraphe [17].

There are a number of other authoring environments for education and industry. VIPLE is a visual programming environment whose design addresses easy programming of IoT and RaaS (Robot as a Service) systems in education [5]. Authoring tools such as RoboStudio also address easy interactive programming of robots in home and service [7].

Current research in embodied computing thus serves to underscore the importance of constructivist practices centered around software to represent physical states in hardware.

Real-Time Programming

Authoring tools can be used as aids in project development, but their effects on these projects may only be seen upon completion, or when the project is run separately. *Real time* authoring tools display their effects so that the author can see them as they make modifications with the tool. Real-time authoring tools thus allow for shorter development time by reducing effort to separately prepare and execute a project.

Wang and Cook [20] describes creating a framework based on the idea of *on the fly programming*, which enables codes to be edited complied, and precisely integrated into programs while it is running. In this sense, code becomes an "expressive instrument" in real time. This framework is based on the programming language ChucK, a "concurrent, on-the-fly audio programming language," which has several features that support *on the fly programming* [20]. Ruru is an example of such a visual environment that handles real-time authoring by allowing the users to see robot input then tune parameters as the robot is executing a physical behavior [8].

Programming by Demonstration

Another potentially less technical method of authoring robot behaviors is "Programming by Demonstration," defined as introducing new behavior to a system by showing the system it in the form of examples [3]. A main goal of such a system is to reduce the need of programming ability in adding functionality or creating prototypes. These systems take a much more physical approach to authoring than our proposed verbal tool.

Zöllner, Asfour, and Dillmann divide the programming by demonstration cycle into three main phases: perception and interpretation of the demonstrated task, generalizing the perceptions into abstract tasks, and mapping the abstract tasks to specific robot targets [21]. These steps make it clear that there is no real time aspect to programming by demonstration. The results of demonstrating even a single action can only be seen in programming by demonstration systems until the third phase [21].

These systems rely heavily on sensors to provide data for the robot, such as cameras and sensor gloves for the system described in Zöllner, Asfour, and Dillmann's paper, and a mouse input device with sensors for the system in Freidrich et al's paper. Simple tasks, such as opening a jar, require several levels of data analysis by the system, breaking the task into several core movements that the system can learn, and eventually imitate [21].

A more recent and potentially more robust version of a programming by demonstration system is presented in a paper by Alexandrova et al, where the user demonstrates the task only once to the robot. The single demonstration, coupled with a dialog system and graphical user interface can give the same or better levels of success that these previous multiple demonstration systems achieved [1, 3, 21]. The tasks chosen to evaluate the system were one and two handed picking up of objects, as well as organization of multiple objects into different boxes. The reported user completion and perceived difficulty of these tasks with the system provided positive results in terms of this paper. However, its use of a both a dialog system and graphical user interface show that it is not solely a programming by demonstration system. Its use of a dialog system furthers the idea that our dialog based authoring tool may be an effective system [1].

Authoring Tools for Robot Behaviors

Prior research includes various approaches to specifying social behaviors for robots, including visual programming environments [9, 18], the use of tangible blocks [19], and dialogue-based authoring [11].

Visual programming languages offer the ability to manage large or complex tasks. Because behaviors can be easily be visualized on a screen, users may be able to manage a more complex behavior. Non-technical users may find the graphical visualization of a task or behavior especially helpful [18]. During execution of an authored program, a visual programming language also is able to notify authors when certain events occur [9] so that debugging becomes a more streamlined process. However, there is often a large learning curve associated with a large or complex visual programming language, which is an obstacle to rapid prototyping for novice users.

Tangible blocks are very intuitive to use, as the authors were able to create scenarios without any kind of instruction. However, this easy to use approach is very limited in the kinds of behaviors that the author is able to express through the robot. Any kind of conditional or iterative behavior cannot be created using tangible blocks, which prevents a robot programmed using it from executing any kind of multi-branched behavior [19].

We believe that the dialogue based tools are the most promising for rapid prototyping and programming for end users where no learning is required. Although the dialogue based approaches are promising, prior systems have been complex, requiring a significant amount of background from users. Gorostiza and Salichs [11] present a Natural Programming System (NPS), which takes in verbal input from an author, and then groups into units known as "skills," which implement "actions" and "conditions." However, the author can open and close branches of execution, and has to refer back to branches, which is shown whenever a scenario has multiple choices. The author, having a "depth-first approach" in which they completely author one possible outcome, must tell the robot to refer back to the original branch. A limitation in the NPS is that the author must figure out the most efficient way to create a given scenario. We address this issue by having a limited set of utterance patterns in our system, removing any kind of guesswork for the author [11].

The increasing interest in authoring tools for social robots and the limitations of prior work point toward a need for tools that offer natural ways of rapidly authoring social interactions for robots. In the next section, we describe our design goals for addressing this problem.

DESIGN GOALS

Based on prior work on interaction authoring, particularly the limitations of alternative approaches to authoring social behaviors for robots, and iterative design sessions that involved bodystorming and paper prototyping, we have developed a set of design requirements for a verbal authoring approach that would support the end-user rapid prototyping of robot behaviors with no learning:

- During the authoring phase, authors must be able to iteratively modify a behavior by appending its smallest unit and keep track of their "growing" behavior.
- Variability in speech recognition accuracy and speech errors made by the author must be accommodated by our system.
- From our pilot tests, we found that authors conditionally affecting the robot's attention is crucial to give authors time to prepare speech and reduce stress.
- Confirmation or cancellation of every unit behavior that is authored will give the author a second layer of validation.
- The system should easily allow authors to keep track of their growing behavior and to start the behavior from scratch.
- Finally, authors must be able to periodically evaluate the current behavior in order to ensure the interaction is accurate.

In the next section, we describe the design and implementation of an authoring and simulation system that meets these design requirements.

SYSTEM

System Implementation

Our implementation of the components described below used Softbank Robotics's Nao robot for speech and execution and Google's Speech Recognition API for speech recognition. Our project was coded in Python 2.7 and used the NaoQi module for communication with the Nao and the `SpeechRecognition` module for interfacing with Google's Speech Recognition API.

Verbal Domain

The planned system must be able to take speech and fit it with a possible behavior pattern comprising of phrases and phrase templates, we call this set of tokens a *verbal domain* because the author must be able to say any combination of these tokens out loud. We restrict which words can be in the verbal domain; these are discussed in the *dialogue* section. We categorize each token as a *pattern*, *event*, *action*, or *memory*. Patterns are templates that specify logic control. For example, there may be a pattern that waits until a certain event becomes true, then executes a certain action. Events are either true or false and are dependent on the robot's perceived state of the environment. For instance, an event called *person_appears* could be defined to be true only when the robot's facial recognition feature detects a face. Actions are a group of commands that the robot can execute, saying a greeting, for example. Lastly, memory tokens are either pre-defined read-only variables or empty and mutable variables, of a string or float type. For specificity, phrase templates are defined to fit tokens from a specified group for each placeholder. The pattern *wait_until_[e]_then_[a]* specifies that only when an event *e* is true, then an action *a* will commence. The code-level implementations of tokens are abstracted away and do not concern the author. Finally, there must be a one-to-one mapping between tokens defined as strings, and tokens defined as Python function and variable addresses so that the composite behavior can be called in execution once the author's speech is parsed and built into said behavior.

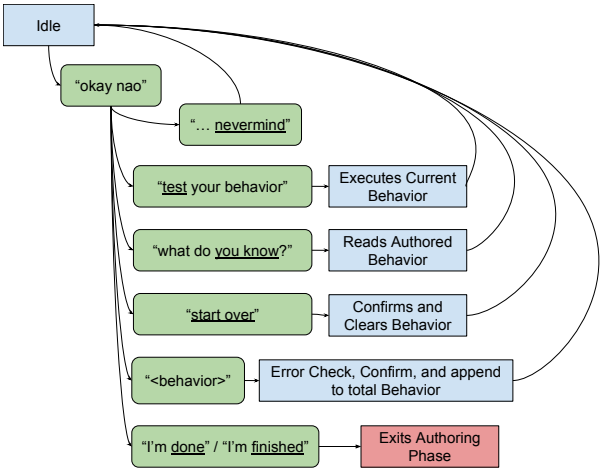

Figure 2. Dialogue flow during the authoring phase. With this constraint, the author appends to a total behavior by authoring and confirming small units of behaviors at a time. The author can keep track of the total behavior by asking "what do you know." For simplicity Nao speech and visual cues, such as eyes lighting green indicating that Nao is listening, is not shown in the diagram.

Dialogue

We constructed the interaction between the author and the Nao to be centered around author readiness. The Nao will only engage in conversation if the author initiates it with the key phrase "Okay Nao." There is an order of precedence associated with certain keywords, some of which are underlined in Figure 3. If no keywords are found, then the system interprets that as an author-intended behavior. Otherwise, the robot will return to idling if the author says "nevermind" in the phrase, say the current behavior if "you know" (as in "what do you know") is in the phrase, delete the current behavior if "start over" is in the phrase, execute the current behavior if "test" is in the phrase, and finally, exit the authoring phase if "done" or "finished" is in the phrase. Except during idling, the Nao will light its eyes and chest lights green when the microphone is recording audio, so that visual cues are present to facilitate smooth dialogue between the author and Nao. Otherwise, an author may unknowingly have a leading portion of their speech cropped off without visual cues. Lights may serve as primitive body language, which is crucial in person-to-person conversation. In this case, we are using the Nao's body lights to play an analogous role to that of physical gestures, which may increase human engagement in the interaction [15].

Once the author says a desired behavior, the speech recognition component returns several possible transcripts of the author's speech, ranked by probability. Each transcript is parsed for syntactical correctness, and a valid transcript will be confirmed with the author. Syntactical correctness means that an arrangement of tokens in the verbal domain can be found so that every token as an argument to a token template has the correct type; e.g., action tokens are in template placeholders where an *[a]* is specified, event tokens are in template placeholders where an *[e]* is specified, etc. Choosing from a list of multiple transcripts is important due to common substitutions of different forms of the same word by Google Speech API. For example, "giving" may be heard and transcribed as "given", and "to" as "two" in the transcript with the highest confidence of accuracy. If the author's transcript is not valid, then the author will be notified. If one is found, then the behavioral unit that is parsed will be appended to a total behavior. We discuss "validity" of transcripts below.

Given a verbal domain, there may be multiple, a single, or no possible arrangements of tokens to fit a given speech transcription. This makes any transcription valid in multiple ways, uniquely valid, or invalid, respectively. To find a fitting of a transcript, we made an algorithm to generate a tree where each node is one possible fitting of arguments to it's parent node (Figure 3). If no possible argument fittings are found for a node, then that node is deleted from the tree. Each branch of the tree is grown until the base case: the only remaining leaf nodes are found to be standalone tokens, i.e. tokens without arguments. The end result is a tree of possible ways tokens from the verbal domain can fit a given speech transcript.

Execution

Once a total behavior comprised procedurally of behavior units have been authored to the author's satisfaction, execution of the interactive behavior starts. There is a one-to-one mapping between tokens in the verbal domain and the addresses of their corresponding python definitions. This mapping allows an easy way to execute a given behavior when the comprising tokens are defined in Python. Arguments of token functions must be interpreted recursively since the depth of the authored behavior is arbitrary, so we define and pass another tree representation of arguments into each function shown in Figure 4.

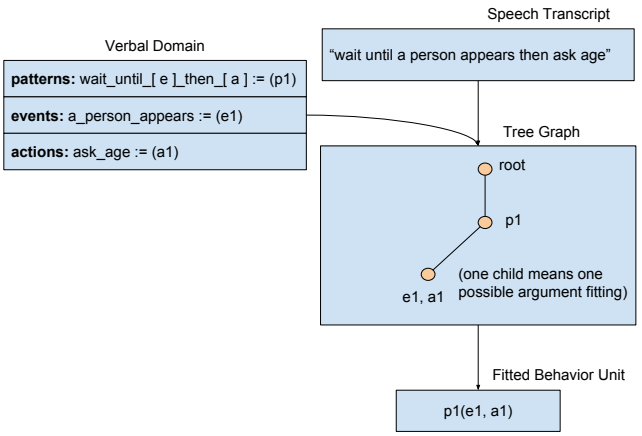

Figure 3. Algorithm to fit behavior to speech transcript. A tree graph is recursively generated to find all possible fittings using tokens from the verbal domain.

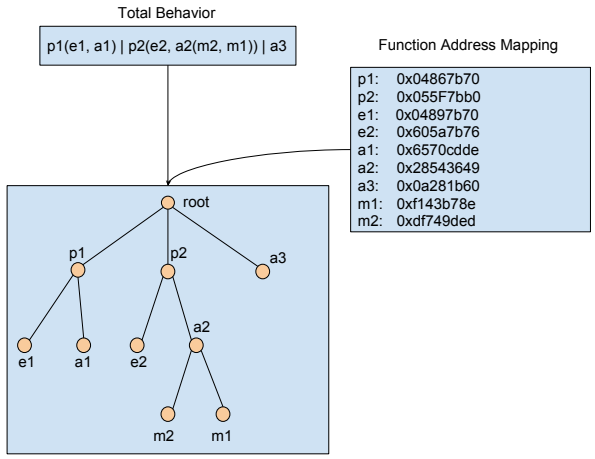

Figure 4. Execution model. A tree graph of function and variable addresses is built from a completed total behavior. Then the tree is evaluated depth-first, unless the function implementation in Python evaluates its arguments in a different order.

USER STUDY

To assess the usability and effectiveness of our proposed approach for authoring human-robot interactions, we designed and carried out a user study in which participants were asked to design applications for a set of given scenarios. We hypothesized that *verbal* authoring will improve the effectiveness of the authoring process, across measures of usability, task load, and user performance, compared to the state-of-the-art *graphical* authoring approaches.

Participants

Our study recruited 25 participants from the university campus community who were native English speakers between the age of 18 to 65, without restrictions of majors or prior knowledge in programming. Of the 25 recruited participants, 24 were able to complete the experiment and are represented in our data. For personal reasons, one of our participants left before completing any part of the study, and is therefore not represented in the data shown in this paper.

Study Design

Our study followed a 2×1 between-participants design in which the authoring method was manipulated to be either *verbal* or *graphical*. Participants were randomly assigned to one of these experiment conditions. Twelve participants were assigned to the verbal condition where they conducted authoring tasks using our verbal-authoring approach, as shown in Figure 5, and 12 participants were assigned to the graphical condition to use Choregraphe as an interface for authoring tasks.

Study Tasks

We created four authoring tasks, which we will continue to refer to as *scenarios* or desired robot behaviors, to be used with verbal authoring and graphical authoring. Because we hypothesized that the benefit of a verbal authoring tool over a graphical authoring tool would be present for specifically simple behaviors, the scenarios presented in our study were relatively simple. For each scenario we created an associated verbal domain for the verbal condition and a list of suggested *boxesr* for the graphical condition, which will be explained below. The order in which the four scenarios were to be completed were also randomized for each participant. The four scenarios are described below:

- Scenario A
    1. The robot should wait until it sees someone then greet them and talk about a sale.
- Scenario B
    1. The robot should wait until a person appears then ask the person for their name.
    2. The robot should then wait until the person gives their name then store the response in memory.
    3. Finally, the robot should tell the user about a sale and say goodbye.
- Scenario C
    1. The robot should wait until a person appears then ask if they are under ten.
    2. The robot should wait until the person confirms their age then store the response in memory.
    3. The robot should instruct the person to go down the ramp if they are younger than ten.
    4. The robot should instruct the person to go down the stairs if they are older than ten.
- Scenario D
    1. The robot should wait until a student appears then ask the student for the number of the question with which they need help.
    2. The robot should wait until the student gives the question number then store the response in memory.
    3. The robot should finally answer for whichever question, either one or two, the student needs help.

Study Procedure

Tutorial Session. The participants first watched an instruction video on the system they were assigned to use during the experiment. The graphical condition participants watched a 13-minute tutorial video on the basic functionality of Choregraphe. The tutorial introduced participants to the set of *behavior boxes* they would use during their tasks. Participants in the verbal condition watched a 12-minute tutorial video on usage and items for the verbal authoring tool, including the verbal domain and key phrases outlined in Figure 3. In both of the tutorial videos, an example scenario was authored. After all participants watched the videos, they were given 20 minutes to practice the same example scenario. During practice, experimenters gave clarification and/or assistance necessary to help participants learn the authoring system and complete the example scenario if possible during the allotted time frame.

Experimental Session. After the successful completion of the example scenario or 20 minutes, whichever came first, participants conducted and tested four experiment scenarios in a randomized order, with a maximum time slot of 20 minutes allotted for each scenario. All participants were given the description of their current scenario broken down into list of steps. Participants in the verbal condition were given the verbal domain and those in the graphical condition were given list of suggested behavior boxes. Scenario descriptions, verbal domains, and suggested behavior boxes were all given on paper.

Measurement

For the duration of the tutorial and experimental session, participants were video-recorded for their facial expressions and words said to capture where during the authoring they were confused and frustrated. In addition, the computer was screen-captured in the graphical condition to record how participants authored behaviors.

Task duration and levels of task completion were recorded. After the experimental session, participants completed System Usability Scale (SUS) and NASA Task Load Index (NASA-TLX) surveys, which are standardized scales for usability [4] and mental effort, respectively. The experimenters then conducted a semi-structured interview with four questions on the experience of participants with their assigned authoring tool. Participant were given compensation based on an hourly rate of $10 USD rounded up by 30 minutes.

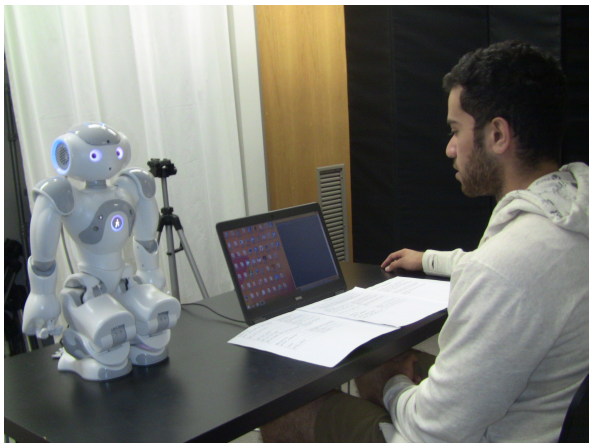

Figure 5. The setup of our experiment under the verbal condition. The author (right) interacts with the Nao (left) by speaking into the laptop microphone. The verbal domain and scenario description pages are provided to the author.

RESULTS

MANOVA

Data analysis was performed using multivariate analysis of variance (MANOVA) to control family-wise error rates. MANOVA results showed a significant multivariate effect for the dependent variables in a group in relation to authoring interface, $F(1,22) = 6.0044$, $p = .001472$. The between associations among dependent variables found that the differences of SUS scores, task completion, and task duration between verbal condition and graphical condition were significant, while that of the NASA Task Load Index was not. Most variables passed the Shapiro test for normality except for the task duration of Scenario A except for task duration time of Scenario A and Scenario B, all other variables passed the Bartlett test for homoscedasticity. Since we know that MANOVA is fairly robust to deviation from normality and homoscedasticity, especially when the sample sizes are equal, we can conclude that our data meets the assumption for MANOVA.

SUS Score. Following the guidelines suggested by Brooke [4], a SUS score from 0 to 100 was calculated by adding the score contribution of each item ranging from 0 to 4 and multiplying the sum of scores by 2.5. The SUS score for graphical condition ranged from 5 to 65 ($M = 40.62$, $SD = 16.07$). The SUS score for verbal condition ranged from 35 to 90 ($M = 62.5$, $SD = 27.5$). On average, the scores for both systems were below the SUS score of 70 that is considered to be a minimum for interfaces with "good" usability, according to the established guidelines [2]. A t-test was conducted to compare the usability of the two systems. Our analysis found a significant effect of authoring interface on the SUS score, $F(1,22) = 9.193$, $p = .006123$. As shown in Figure 6, the SUS score for the verbal condition was significantly higher than that of the graphical condition.

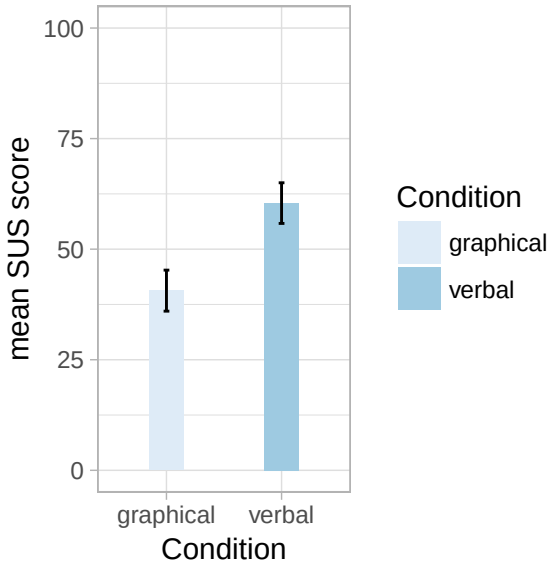

Figure 6. System Usability Score (SUS) across graphic and verbal authoring conditions.

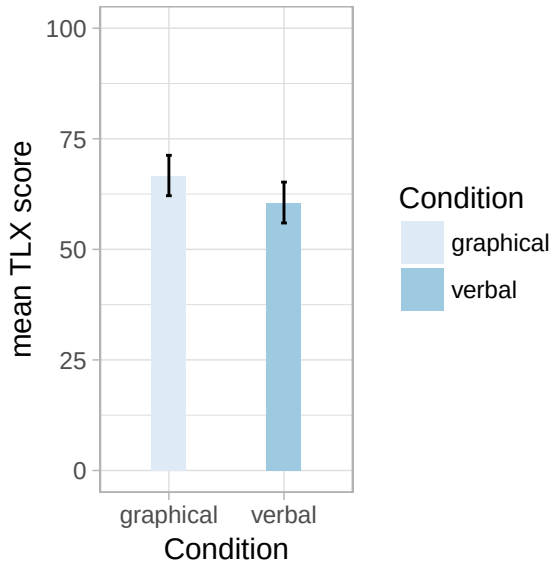

Figure 7. NASA Task Load Index (TLX) scores across graphic and verbal authoring conditions.

Task Load. Weighted NASA-TLX scores were obtained from the calculated results in the NASA-TLX iOS application released by NASA. The NASA-TLX (Figure 7) scores for the verbal condition ranged from 21 to 73.67 ($M = 66.69$, $SD = 15.81$), while those for the graphical condition ranged from 34.67 to 88.33 ($M = 60.58$, $SD = 16.01$). The difference in the scores between the two authoring conditions was not significant, $F(1,22) = .8854$, $p = .357$.

User Performance. Task completion (Figure 8) was calculated by dividing the number of completed steps by the

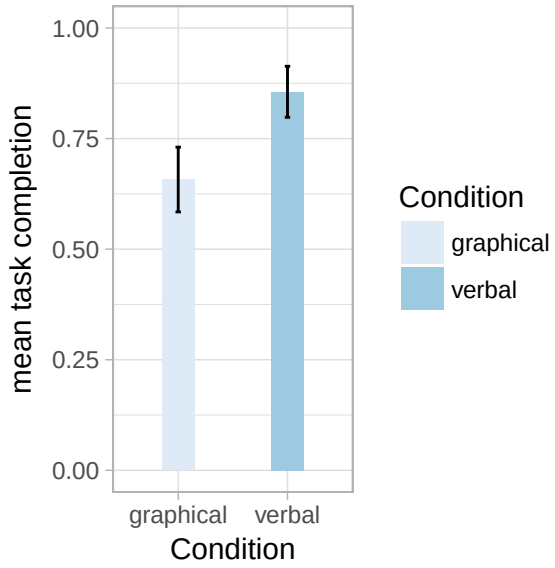

Figure 8. Task completion rate across graphic and verbal authoring conditions.

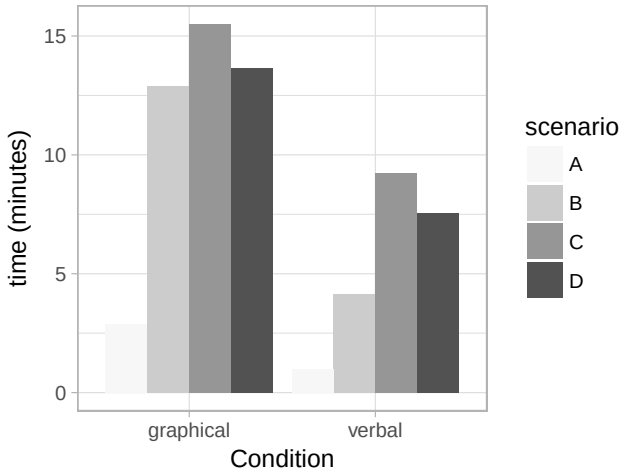

Figure 9. Completion times for all tasks in the graphical and verbal authoring conditions.

total number of steps for each authoring task. Compared to the steps completed in the graphical condition ($M = 0.66$, $SD = 0.25$), the number of steps completed in the verbal condition ($M = 0.86$, $SD = 0.20$) was significantly higher, $F(1,22) = 4.5296$, $p = 0.04476$.

In addition, task completion time for each scenario was recorded in minutes. On average, task completion time for all four scenarios was significantly shorter than that of graphical condition. As shown in Figure 9, scenario A was in both conditions the shortest in task duration. In the graphical condition, participants took significantly longer to finish scenario A ($M = 2.86$, $SD = 2.50$) than those in the verbal condition did ($M = .98$, $SD = 0.27$), $F(1,22) = 6.7602$, $p = .01634$, longer to finish scenario B ($M = 12.88$, $SD = 6.18$) than those in the verbal condition ($M = 4.15$, $SD = 2.48$), $F(1,22) = 20.615$, $p = .0001612$, longer to finish scenario C ($M = 15.5$, $SD = 5.38$) than those in the verbal condition ($M = 9.24$, $SD = 5.3$), $F(1,22) = 8.2496$, $p = .008851$, longer for scenario D ($M = 13.64$, $SD = 5.38$) than those in the verbal condition ($M = 7.54$, $SD = 5.3$), $F(1,22) = 5.6206$, $p = 0.02693$.

DISCUSSION

Our results partially support our hypothesis; the usability of both the verbal authoring tool and Choregraphe were below the threshold of "good" usability, according to the established guideline about SUS score [2]. However, the SUS score usability of the verbal authoring tool was rated significantly higher than that of Choregraphe. The user performance under the verbal condition was significantly higher than the graphical condition in terms of the average number of steps completed for each scenario. Authoring with the verbal condition was therefore on average a more effective authoring tool in completing the provided scenarios than authoring with the graphical condition. In addition, the mean task duration time was significantly shorter in the verbal tool than that in Choregraphe across all four scenarios. In other words, the authoring tasks took much less time when the verbal tool was used. We noted that for both tools, the mean completion time had the same relative rank. Scenario C had the longest average completion time, followed by scenario D, scenario B, then finally scenario A. The ranking of task completion time indicated that the relative complexity of each scenario was preserved under either condition. Contrary to our hypothesis, there was no significant difference between the perceived work load of authoring experience with verbal authoring and that with Choregraphe.

Limitation and Future Work

During the design process, we found that our use of speech recognition made some choices of phrases and phrase templates more practical than others. Fixes such as adding software heuristics and replacing words that were frequently transcribed incorrectly facilitated a smoother authoring process. However, such changes cannot be practically sustained for creating more complex scenarios. Thus, future work may investigate whether some more mature speech recognition process could be implemented to recognize wider range of phrases. Furthermore, speech parsing in our implementation was close to literal, whereas in the future, a system that is more suggestive and flexible towards the author's speech errors may be adopted. For example, when presented with an invalid transcript, the Nao may suggest an alternate, valid transcript with *"did you mean ... ?"* In addition, the authoring scenarios designed in this project were limited to only speech interaction, but the physical capabilities of Nao and other robots could enable future verbal authoring design to incorporate more physical activities (e.g. handing the participant a ball). Future exploration might seek to compare authoring experience of participants with varying level of programming ability, which might cause difference in their experience in navigating the system.

Finally, we also considered whether an *edit* feature to change a specific behavioral unit in a large built behavior would violate the core tenants of our verbal authoring tool as a *conversational* tool rather than a "graphical editor without the screen." Creating an edit feature might cause our system to closely resemble a spoken programming language, rather than the natural language tool that we wanted to create.

CONCLUSION

Robotic products are becoming increasingly commonplace. Introducing these products into day-to-day settings involves an interaction designer to "author" the robot's behavior. Traditionally, the designer authors interactions using connected visual nodes that model behaviors. Visual authoring enables designers with little programming experience to design robot behaviors but still requires the designer to learn the environment's conventions and the setup of the environment-robot system, which can become barriers to rapid prototyping and design.

In this paper, we presented an authoring approach that aims to remove these barriers and provides the designer with the ability to create and implement robot behaviors using natural spoken language. We implemented our verbal authoring tool and evaluated the effectiveness of and user experience with the tool. In the user study, participants used either verbal authoring or Choreographe, a graphical authoring tool, to complete four

different authoring scenarios. We measured mean completion time for authoring tasks and had participants rate the authoring tools using the SUS scale, a subjective measure of usability, and the TLX score, a measure of perceived work load.

Based on data from 24 participants, our findings show our verbal-authoring approach to improve usability and task performance in measures of task-completion rate and task time, without increasing user task load compared to state-of-the-art graphical approach. Our findings highlight the promise of verbal authoring to serve as a usable and effective approach to end-user rapid prototyping and programming of social robots that requires little training or prior knowledge.

ACKNOWLEDGMENTS

Removed for blind review.

REFERENCES

1. Sonya Alexandrova, Maya Cakmak, Kaijen Hsiao, and Leila Takayama. 2014. Robot programming by demonstration with interactive action visualizations.. In *Robotics: science and systems*. Citeseer.
2. Aaron Bangor, Philip T Kortum, and James T Miller. 2008. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction* 24, 6 (2008), 574–594.
3. Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Robot programming by demonstration. *Springer handbook of robotics* (2008), 1371–1394.
4. John Brooke and others. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996), 4–7.
5. Yinong Chen and Gennaro De Luca. 2016. VIPLE: visual IoT/robotics programming language environment for computer science education. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 963–971.
6. Michael A Cohen, Todd S Horowitz, and Jeremy M Wolfe. 2009. Auditory recognition memory is inferior to visual recognition memory. *Proceedings of the National Academy of Sciences* 106, 14 (2009), 6008–6010.
7. Chandan Datta, Chandimal Jayawardena, I Han Kuo, and Bruce A MacDonald. 2012. RoboStudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2352–2357.
8. James P Diprose, Bruce A MacDonald, and John G Hosking. 2011. Ruru: A spatial and interactive visual programming language for novice robot programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*. IEEE, 25–32.
9. D Glas, T. Kanda, and H. Ishiguro. 2016. Human-robot interaction design using Interaction Composer eight years of lessons learned. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 303–310.
10. D Glas, Satoru Satake, Takayuki Kanda, and Norihiro Hagita. 2012. An interaction design framework for social robots. In *Robotics: Science and Systems*, Vol. 7. 89.
11. Javi F Gorostiza and Miguel A Salichs. 2011. End-user programming of a social robot by dialog. *Robotics and Autonomous Systems* 59, 12 (2011), 1102–1114.
12. Mark Green. 2003. Towards virtual environment authoring tools for content developers. In *Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, 117–123.
13. Björn Hartmann, Scott R Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. ACM, 299–308.
14. Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, 91–100.
15. Derek McColl and Goldie Nejat. 2014. Recognizing emotional body language displayed by a human-like social robot. *International Journal of Social Robotics* 6, 2 (2014), 261–280.
16. Monica N Nicolescu and Maja J Mataric. 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM, 241–248.
17. Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*. IEEE, 46–51.
18. Allison Sauppé and Bilge Mutlu. 2014. Design patterns for exploring and prototyping human-robot interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1439–1448.
19. 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.
20. Ge Wang and Perry R Cook. 2004. On-the-fly programming: using code as an expressive musical instrument. In *Proceedings of the 2004 conference on New interfaces for musical expression*. National University of Singapore, 138–143.

21. R Zollner, Tamim Asfour, and Rüdiger Dillmann. 2004. Programming by demonstration: Dual-arm manipulation tasks for humanoid robots. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, Vol. 1. IEEE, 479–484.