FLEX-SDK: An Open-Source Software Development Kit for Creating Social Robots

Patrícia Alves-Oliveira[*]
University of Washington
Seattle, Washington, USA
patri@cs.washington.edu

Kai Mihata[*]
University of Washington
Seattle, Washington, USA
kaim2@cs.washington.edu

Raida Karim
University of Washington
Seattle, Washington, USA
rk1997@cs.washington.edu

Elin Björling
University of Washington
Seattle, Washington, USA
bjorling@uw.edu

Maya Cakmak
University of Washington
Seattle, Washington, USA
mcakmak@cs.washington.edu

ABSTRACT

We present FLEX-SDK: an open-source software development kit that allows creating a social robot from two simple tablet screens. FLEX-SDK involves tools for designing the robot face and its facial expressions, creating screens for input/output interactions, controlling the robot through a Wizard-of-Oz interface, and scripting autonomous interactions through a simple text-based programming interface. We demonstrate how this system can be used to replicate an interaction study and we present nine case studies involving controlled experiments, observational studies, participatory design sessions, and outreach activities in which our tools were used by researchers and participants to create and interact with social robots. We discuss common observations and lessons learned from these case studies. Our work demonstrates the potential of FLEX-SDK to lower the barrier to entry for Human-Robot Interaction research.

CCS CONCEPTS

• **Human-centered computing → User interface programming**.

KEYWORDS

Human-robot interaction, end-user programming, personalization

ACM Reference Format:
Patrícia Alves-Oliveira, Kai Mihata, Raida Karim, Elin Björling, and Maya Cakmak. 2022. FLEX-SDK: An Open-Source Software Development Kit for Creating Social Robots. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3526113.3545707

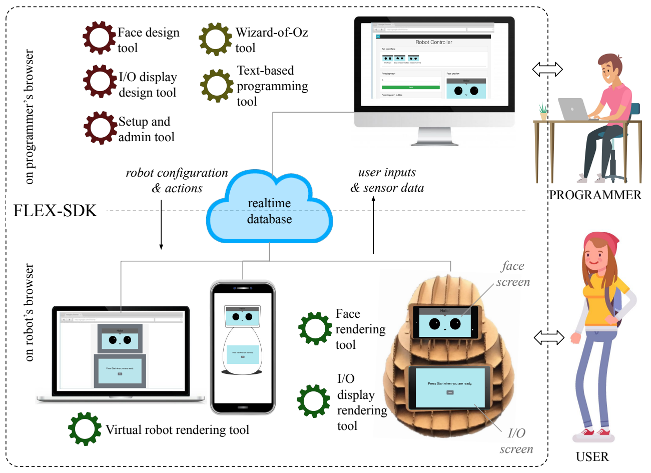


Figure 1: FLEX-SDK consists of browser-based tools for (i) creating interaction content for a social robot, (ii) rendering such content on the robot, (iii) direct user control of the robot (Wizard-of-Oz), and (iv) programming the robot to autonomously interact with users.

1 INTRODUCTION

Social robots have become increasingly ubiquitous in a wide range of applications from healthcare to education. A number of large companies and start-ups have recently created impressive social robot products, while more research platforms have become available at lower price points. Despite this rapid growth, social robots are still far from being accessible to anyone who can think of a good application for them. Creating a social robot application currently requires committing to a robot platform with a specific form factor and learning robot-specific programming tools. These tools often require software development expertise and restrict the level of customization possible. As a result, the community of social robot application developers is still small and exclusive, and the variation of social robots used in research and practice is limited [6]. Our goal is to address this problem.

We aim to enable anyone who can envision a social robot to be able to create it without having to purchase a specific hardware platform and without having to install and learn complex software frameworks. To that end, we developed FLEX-SDK: an open-source software development kit for creating social robots. FLEX-SDK involves tools for (i) creating interaction content for a social robot, (ii)

[*]Both authors contributed equally to this research.

rendering that content on a social robot, (iii) controlling the robot through a Wizard-of-Oz style interface, and (iv) programming the robot to operate autonomously. Although FLEX-SDK can be used with many different types of social robots, its canonical use assumes that the robot has one screen for displaying the robot's face and another touch-screen for input/output interactions with the user.

In this paper we first describe the system and its implementation, and we demonstrate how it can be used to create different social robots and human-robot interactions. We then present nine case studies in which FLEX-SDK was used to create robots as part of an HRI research and outreach. We distill some observations across the different case studies and discuss the potential impact of FLEX-SDK if adopted by the Human-Robot Interaction (HRI) community.

2 RELATED WORK

In this section, we review the different efforts made within industry and research labs to make programming accessible.

2.1 Programming Commercial Robots

All programmable social robots on the market come with a software development kit. SoftBank's NAO[1] and Pepper[2], two of the most widely used social robots, are programmed through a software tool called Choregraphe [24], which allows block-based programming primarily focused on moving the robot through different poses and is extensible through a Python application programming interface (API) and a custom dialog specification language. The small Cozmo[3] robot has a Python based API with fixed interaction content (e.g., facial expressions, expressive movement animations). Misty[4], which is advertised a robot development platform, has a JavaScript SDK involving a suite of tools similar to the ones in FLEX-SDK, such as a browser based command center, API explorer, and skill runner.

2.2 Programming Social Robots

Several efforts within the HRI research community share our goal of making it easier to program social robots and have contributed new systems for programming existing root platforms. For example, Lourens et al. developed TiViPE to enable rapid programming of the NAO robot for robot-assisted autism therapy [19, 20]. Rietz et al. created an accessible interface only for the Pepper robot, enabling non-programmers to conduct WoZ experiments [25]. Datta et al. created RoboStudio for programming interactions with a health-care robot [11]. Interaction Composer [13, 14] is a system with a flow-based interface for programming HRI applications on the social mobile Robovie platform. Saupe et al. developed Interaction Blocks [26], also for the NAO robot, to provide a faster way of designing human-robot interactions based on common interaction patterns. The RoVer system by Porfirio et al. build on this work to automatically verify that an interaction composed of interation blocks satisfies selected interaction norms, such as "the robot should not interrupt the human's speech while the human has the speaking floor" [23]. Another recent system called Interaction Flow Editor

[1]NAO Robot: https://www.softbankrobotics.com/emea/en/nao
[2]Pepper Robot: https://www.softbankrobotics.com/emea/en/pepper
[3]Cozmo Robot: https://ankicozmorobot.com/
[4]Misty Robot: https://www.mistyrobotics.com/

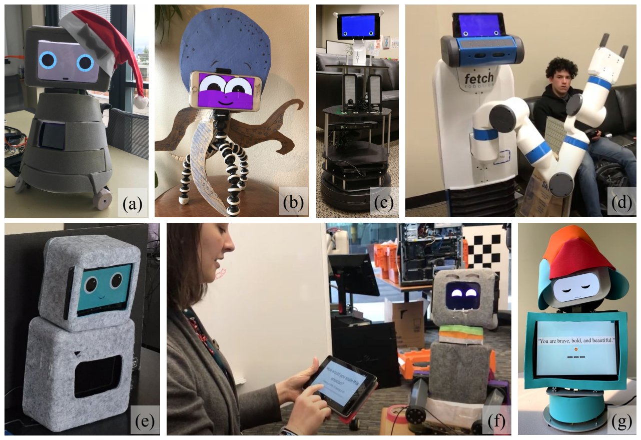


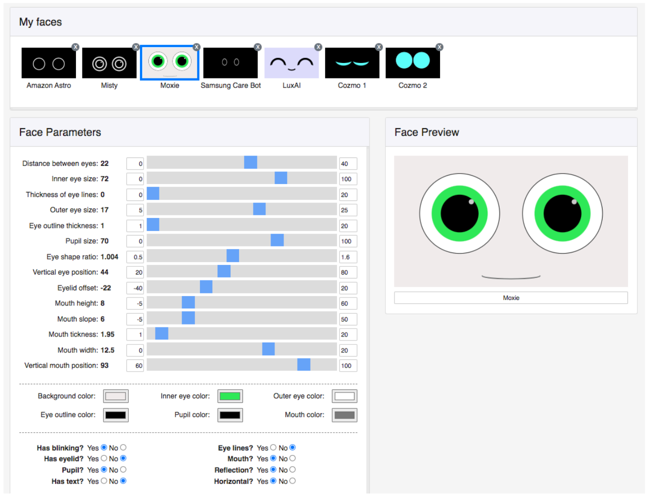
Figure 2: Examples of robots that were created or augmented with FLEX-SDK. The top row shows robots that have only one screen used as a face: (a) and (b) are non-actuated robots, while (c) is an existing mobile robot platform (Turtlebot 2) and (d) is an existing mobile manipulation platform (Fetch) augmented with a simple social face for human interactions. The bottom row has robots with two screens: (e) and (g) have the second screen attached to the robot, while (f) has the user holding it. While (e) and (f) are non-actuated, (g) has a 4-degree-of-freedom neck-and-base mechanism that is also controlled through FLEX-SDK.

(IFE) was developed for the Jibo platform and is intended for rapid prototyping of interactions.

Our work shares the goal of these systems and has many shared features, but it is unique in that it is not designed for an existing robot platform but rather for creating a new platform or augmenting an existing one. The explicit separation of content creation and programming is a property of FLEX-SDK that is shared with fewer existing systems.

2.3 Programming Robots

Researchers in the HRI community have also contributed programming systems for robots that are not necessarily social or designed for interaction. The task of programming and testing the robot is itself a key interaction that end-users of functional robots who have varying technical expertise might need to engage in. Such systems include ROS Commander [21], RoboFlow [2], and Code3 [16] designed for the PR2 robot; CoStar [15, 22] designed for a collaborative Kuka industrial robot; and iCustomPrograms designed for Savioke's hotel delivery robot [10, 17]. Other related work includes a WoZ system intended for participatory design of interactions, for capturing data from the operator to learn autonomous programs [27]. A recent literature survey by Ajaykumar et al. provides details on many of these systems as well as a framework for how they can be categorized and evaluated [1]. While the development of programming systems that are not social/interactive is important for certain robotics applications, such as manufacturing, the work we present in this paper targets social robotics.



Distance between eyes: **22**	0		40
Inner eye size: **72**	0		100
Thickness of eye lines: **0**	0		20
Outer eye size: **17**	5		25
Eye outline thickness: **1**	1		20
Pupil size: **70**	0		100
Eye shape ratio: **1.004**	0.5		1.6
Vertical eye position: **44**	20		80
Eyelid offset: **-22**	-40		20
Mouth height: **8**	-5		60
Mouth slope: **6**	-5		50
Mouth tickness: **1.95**	1		20
Mouth width: **12.5**	0		20
Vertical mouth position: **93**	60		100

Background color: Inner eye color: 🟩 Outer eye color:
Eye outline color: ■ Pupil color: ■ Mouth color: ■

Has blinking? [✓ Yes] No
Has eyelid? Yes [✓ No]
Pupil? [✓ Yes] No
Has text? Yes [✓ No]
Eye lines? Yes [✓ No]
Mouth? [✓ Yes] No
Reflection? [✓ Yes] No
Horizontal? [✓ Yes] No

Figure 3: Screenshot of the FaceEdit tool in FLEX-SDK showing the different face attributes (continuous, color, or binary) that can be changed.

3 SYSTEM DESCRIPTION

FLEX-SDK consists of software tools for creating interaction content, rendering the content on the robot, controlling the robot through a Wizard-of-Oz interface, and programming the robot to operate autonomously (see Fig. 1). In this section, we describe the different tools that are part of this system.

3.1 Robot Embodiment

FLEX-SDK is intended for interactive social robots that have a screen-based rendered face, as it provides ways to design digital robot faces and programmatically control it. In addition to the face, the robot may also have a second screen for displaying information and/or getting input from the user. The second screen can either be attached to the robot or external. Figure 2 shows examples of robots with various embodiments that were created and programmed using versions of FLEX-SDK, including ones with one or two tablets with different configurations of the second tablet.

3.2 Content Creation Tools

Rather than providing a fixed set of robot screens or requiring programmers to set the content of screens completely programmatically, FLEX-SDK provides tools for designing two screens (robot face or input/output) through *direct manipulation.*

3.2.1 FaceEdit. The FaceEdit tool in FLEX-SDK allows users to create social robot faces from basic face elements like eyes and mouth. A survey of existing screen-based robot faces by Kalegina et al. identified the different elements that such faces can include, as well as the different parameters of each element that can be varied [18]. Figure 3 shows a screenshot of the FaceEdit tool. The tool has radio buttons for binary parameters (e.g., hasMouth?, hasPupil?) and sliders for continuous parameters (e.g., vertical eye position, distance between eyes, pupil size). In addition, it has color selectors for changing the color of the different elements. The tool includes a preview rendering of the face that changes as the programmer interacts with the attribute controls. The programmer can name the faces they create and browse previously created faces to edit, copy, or delete them through FaceEdit.

With 28 attributes that can be modified, FaceEdit allows creating a wide range of robot faces. To demonstrate this variety, we recreated the faces of six popular social robots that have screen-based robot faces, as shown in Figure 4.

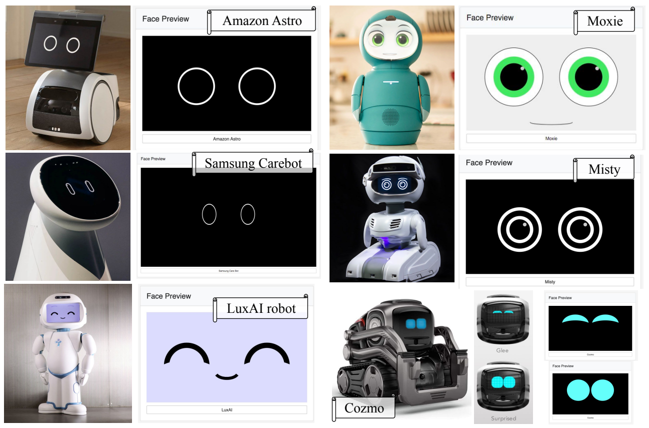


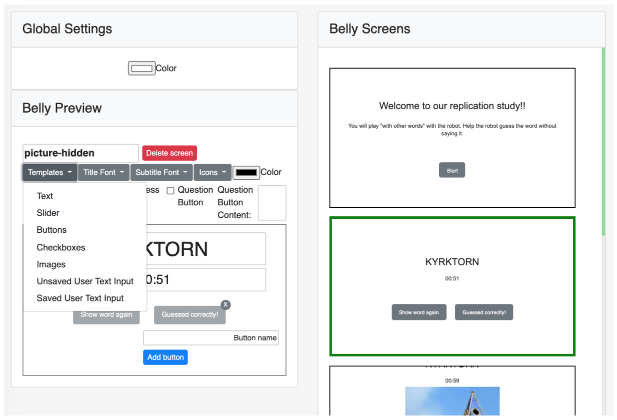
Figure 4: Examples of commercial robots with screen-based faces and faces recreated using the FaceEdit tool in FLEX-SDK, showing the expressivity of our tool.

3.2.2 IOEdit. To enable interactions through a second input/output tablet, FLEX-SDK provides the IOEdit tool which allows designing interaction screens that can involve a combination of large and small text, a slider, an image, buttons, and checkboxes (see Figure 5). The tool provides predefined templates with a different combination of elements. Each element can be directly edited while a rendering of the screen, and the way it would look on the robot screen is updated. The process of creating I/O screens closely resembles the creation of slides (e.g., Powerpoint, Keynote). IOEdit allows the user to name their screens and to browse the different screens they created to edit or delete them.

3.3 Additional Robot Capabilities

3.3.1 Actions. In addition to setting the visual displays, FLEX-SDK enables the use of speakers on the robot screens for making sounds or using text-to-speech to say something. For robots that have additional actuators (e.g., the 4 Degree-of-Freedom robot shown in Fig 2(g)), FLEX-SDK is further extended to provide a control interface, in the same way screen-based actions are controlled, both programmatically and through the WoZ interface (see section 3.7).

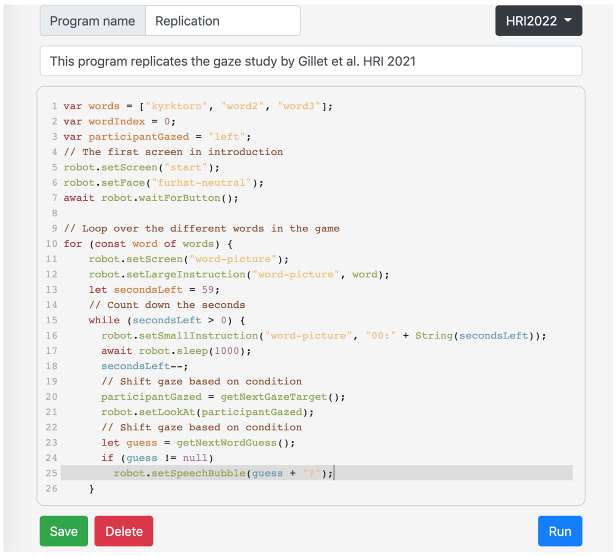
3.3.2 Perception. To enable robust, error-free social interactions with users, FLEX-SDK relies on obtaining user input thorough a touch screen via the input elements added to the I/O screen with IOEdit. In addition, we take advantage of the microphone available on most tablet screens to enable speech input from users. Programmers can define a fixed set of commands that can be recognized and used as part of the robot's interaction.



Welcome to our replication study!!

You will play "with other words" with the robot. Help the robot guess the word without saying it.

Figure 5: Screenshot of the IOEdit tool in FLEX-SDK showing the list of available screen templates.



This program replicates the gaze study by Gillet et al. HRI 2021

```javascript
var words = ["kyrktorn", "word2", "word3"];
var wordIndex = 0;
var participantGazed = "left";
// The first screen in introduction
robot.setScreen("start");
robot.setFace("furhat-neutral");
await robot.waitForButton();

// Loop over the different words in the game
for (const word of words) {
    robot.setScreen("word-picture");
    robot.setLargeInstruction("word-picture", word);
    let secondsLeft = 59;
    // Count down the seconds
    while (secondsLeft > 0) {
      robot.setSmallInstruction("word-picture", "00:" + String(secondsLeft));
      await robot.sleep(1000);
      secondsLeft--;
      // Shift gaze based on condition
      participantGazed = getNextGazeTarget();
      robot.setLookAt(participantGazed);
      // Shift gaze based on condition
      let guess = getNextWordGuess();
      if (guess != null)
        robot.setSpeechBubble(guess + "?");
    }
```

Figure 6: Text-based programming tool in FLEX-SDK.

3.4 Rendering Tools

The robot face and I/O screens created by the programmer are rendered on two separate screens on the robot. The face renderer automatically animates the designed faces by adding natural eye blinking. In addition, it can modify the gaze direction of the robot's eyes by moving the inner part of the eye relative to the outer part. The gaze direction can be set programmatically or through the WoZ interface (see section 3.7).

In addition to the facial features, the face can have text displayed at the top of the screen, appearing like a speech bubble. This text is similarly set programmatically or through the WoZ interface. The face rendering tool also produces the commanded robot sounds and utterances. The I/O screen rendering tool displays the designed I/O screens and detects when a user input is provided through the screen. It receives commands to enable and disable speech recognition and detects when a speech command is received.

In addition to rendering the face and I/O screens on two separate screens on a robot, as in the examples shown in Figure 2, FLEX-SDK provides a virtual social robot by rendering the two screens onto a simple digital robot image (see Figures 1, 13, 8). This acts as a simulation of the social robot and allows easy testing and debugging on a single computer, simply by opening the virtual robot renderer on a separate browser window.

3.5 Application Programming Interface (API)

The combination of capabilities described in the previous sections results in a rich set of functions for creating interactions with social robots created in FLEX-SDK. These functions are collectively referred to the application programming interface (API) of the system. FLEX-SDK has a dynamic API that evolves as the programmer creates additional content for their robot. The basic set of functions in the API includes: setFace(faceIndex), setSpeechBubble(text), setScreen(screenIndex), setLargeInstruction(screenIndex, text), (sliderValue) getSliderValue(), (buttonName) waitForButton(), (commandName) waitForSpeech(), speak(text), playSound(soundIndex), and sleep(duration).

3.6 Programming Tool

The programming tool in FLEX-SDK allows creating text based programs with functions in the API (see section 3.5). The programming language is JavaScript. Although many programs can be a simple scripted sequence of robot commands from the API, allowing the use of a general purpose programming language like JavaScript enables the programmer to create programs with arbitrary complexity. This includes ability to create variables and lists, loops and conditionals, as well as functions that can be key to creating more intricate programs for not just enabling a particular interaction, but also automating parts of an HRI user study. All functions in the API that correspond to robot actions are non-blocking; i.e., the function call in the program only starts the action and does not wait for the action to be complete before moving onto the next line in the program. To capture the end of an action, the program needs to explicitly involve an 'await' command with the associated waiting function, e.g., 'waitForSpeakEnd()'. Event-based user inputs such as pressing a button are similarly captured with 'await' commands.

The programming tool allows the user to browse and copy existing programs, edit their own programs in a syntax-highlighted editor, and run the program on a robot. The tool also has a button "Show robot functions" that displays the robot API with a description of each function and its parameters, including an example. The shown API is automatically adapted to include the user created content (faces, screens, sounds) that has been added onto the robot. A screenshot of the programming tool is shown in Figure 6.

3.7 Wizard-of-Oz Tool

In addition to the programming tool, commands can be issued to the robot directly through the Wizard-of-Oz tool in FLEX-SDK. This tool allows programmers to get familiar with the API and test the content that they have created as part of an interaction. It also enables WoZ studies to be performed before doing any programming or as an alternative when programming is not possible. For instance,

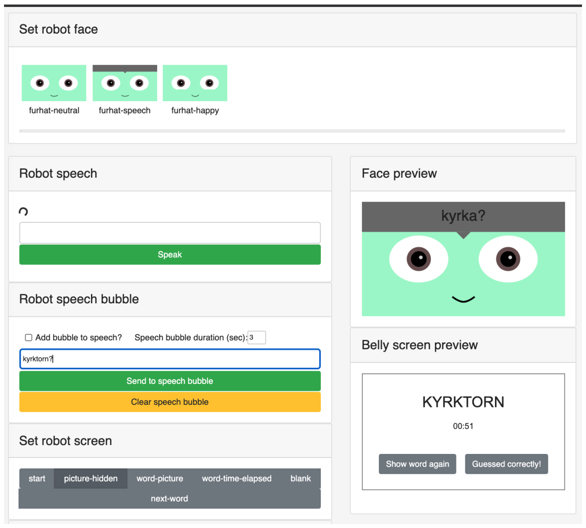


Figure 7: Wizard-of-Oz (WoZ) tool in FLEX-SDK.

if the interaction requires the robot to perceive the environment or understand human speech at a level that is currently not possible to do reliably with autonomy.

The WoZ tool involves different interface elements for commanding the various API functions. For example, it displays the set of faces that have been added to a robot and the operator ("wizard") can set the robot's current face simply by clicking on the desired face. They can set the I/O screen by clicking on a button with the corresponding name for each screen. The operator can also make the robot speak or set the text in the speech bubble on the robot face by typing the text and pressing a button. In addition, the WoZ interface includes buttons for setting the gaze direction of the robot, which has the options none (looking straight), up, down, left, right, and random (periodically changing to look in a different direction). The interface has also been extended to include sliders to control robot joints for the actuated robot neck shown in Figure 2(g). A preview of the robot is shown in the interface for awareness of the operator.

4 DEMONSTRATION

To give a better understanding of how FLEX-SDK can be used to create social robot interactions, we present a walk through of a particular use case. The chosen use case is to create a social robot and program it for a user study replication. In particular, we chose the study by Gillet et al. [12] which won the award for "best HRI user study paper" at the ACM/IEEE International Conference on Human-Robot Interaction (HRI) 2021. The study involves a non-articulated robot head with a back-projected face (Furhat[5] robot). It investigates how the robot's gaze behavior impacts human participation in an interaction involving two humans collaborating in a game. The two participants play the game called "with other words" which involves describing a given word to the robot without saying the word. The robot listens to the humans and tries to guess the word based on what they say. As part of its listening, it shifts its gaze

[5]Furhat Robot: https://furhatrobotics.com/

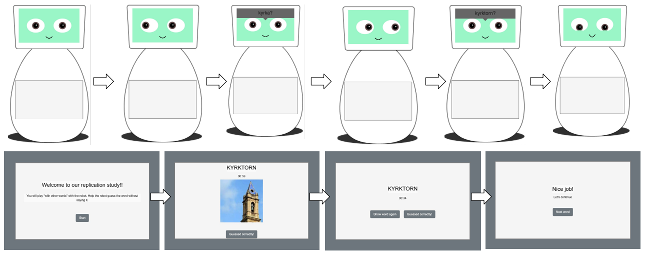

kyrka?
kyrktorn?
Welcome to our replication study!!
You will play "with other words" with the robot. Help the robot guess the word without saying it.
Start
KYRKTORN
00:59
Guessed correctly!
KYRKTORN
00:34
Show word again
Guessed correctly!
Nice job!
Let's continue
Next word

Figure 8: Screenshots of the robot face and user tablet in the replication study demonstration described in section 4.

between the two humans. The study involves manipulating the gaze shifting behavior across the different conditions.

Step 1 - Create robot faces — The key robot actions required for the study are shifting gaze to look at the two participants and speaking to guess the word being described. The face that we designed to replicate the study therefore has large human-like eyes that make gaze shifts noticable (see Figures 7 and 8). This is achieved through the FaceEdit tool by reducing the inner eye parameter which specifies the ratio of the iris relative to the outer eye size. The color of the face is chosen to be similar to the one in the original study and the default facial expression is neutral-to-positive. Two additional faces were created: one with the speech bubble to augment the robot speech when guessing, and another more positive face with a smile and raised cheeks for when the robot correctly guesses the word.

Step 2 - Create I/O screens — The study involved giving participants an iPad which displayed the word that they had to describe to the robot, together with an image of the word that was displayed for a few seconds in case the less proficient speaker did not know the meaning of the word. The image was removed 8 seconds later to avoid distracting participants. The screen also had a timer counting down 60 seconds for each word. We used the robot's I/O screen to enable this interaction. We created different screens for the introduction of the study, for displaying the word with and without the image and for transitioning to the next word after a timeout or correct guess. Examples are included in Figures 5 and 8.

Step 3 - Wizard of Oz test — As in the pilots of the original study, we can recreate the game playing interactions through the WoZ interface once the contents in the first two steps have been created. The screenshot of the WoZ tool shown earlier (see Figure 7) is in fact what it looks like as part of this replication study. For this study the operator would need to use the buttons to set the faces and the different I/O screens, the text entry for guessing the words with speech and the speech bubble, and the gaze direction buttons to shift gaze based on the strategy they are emulating. Despite the rather fast pace of the game our tests indicate that WoZ is feasible for this study with the tool provided in FLEX-SDK.

Step 4 - Programming the interaction — The last step in creating an autonomous interaction is to write a program with the API functions using the programming tool. Figure 6 shows a program created for the replication study, with a

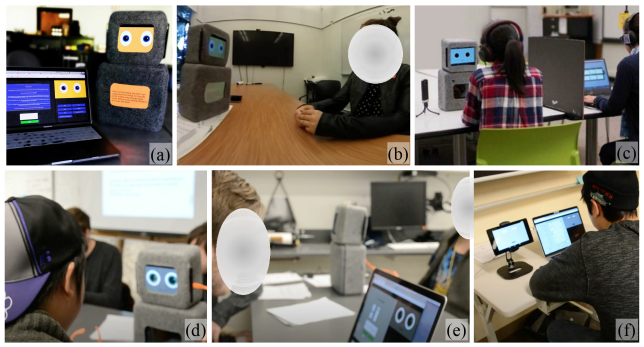


Figure 9: Pictures of robots created with FLEX-SDK being used as part of case studies: (a) Robot and WoZ console for a user study investigating how a social robot's disclosure impacts the participant's response; (b) a participant interacting with the robot in CS-1; (c) a participant interacting with the robot created with FLEX-SDK for a study investigating user preferences for interaction modalities, comparing the physical robot to a robot on a screen and in VR in CS-2; (d) a participant interacting with a robot created with FLEX-SDK; (e) another participant operating the robot through FLEX-SDK WoZ tool in a participatory design study investigating how a robot should respond while listening to a user talk about their stress in CS-3; (f) a participant in CS-3 working with the FaceEdit tool to design a robot face that would feel most attentive and welcoming to them.

few simplifications. In particular, the exact replication of the functions getNextGazeTarget() and getNextWordGuess() would require additional functionalities (e.g., getting speech separately from each speaker, computing distance between words using Word2Vec embeddings) which are possible but beyond the scope of this illustrative demonstration. Instead simpler, randomized versions of these functions were implemented for illustration. Screenshots of the simulated robot and the I/O tablet from running this program are shown in Figure 8, illustrating how the robot can look at the two participants on different sides and how the I/O tablet can guide the participants' interaction as part of the study.

5 SYSTEM IMPLEMENTATION

5.1 Overview

FLEX-SDK is browser based and implemented with HTML, CSS, and JavaScript. The core of the system is a real-time database that contains all information of the users and robots. Each tool is a web page that runs on a browser and communicates with the database to receive data and update the database based on the user or programmer input. The browsers on the robot screens run the two rendering tools, while the browser on the programmer's computer run one of the other tools. All content that is created by the programmer (faces, I/O screens, programs) are pushed to the database and are continuously updated during the editing process. They are initially stored under user data but can be copied over to a particular robot entry in the database through a setup tool.

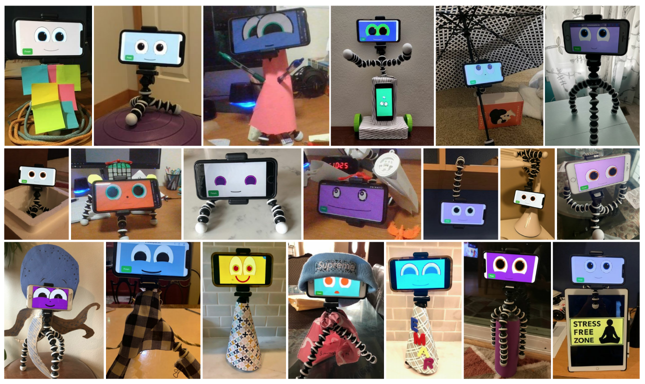


Figure 10: Robots created by participants using the FaceEdit tool in FLEX-SDK as part of CS-4.

In addition to the added content, each robot entry in the database has parameters associated with its current state, e.g., the index of the current face or screen that it is displaying, current gaze direction, or the text currently displayed in its speech bubble. The face and I/O screen rendering tools use this state information as part of its rendering process. When a robot program executes a command or a command is issued through the WoZ interface, the command is transmitted to the robot through these robot state entries in the database. The programming or WoZ tools write onto the realtime database, which immediately pushes this change to the rendering tool, triggering a re-rendering to reflect the state change.

5.2 Implementation Details

FLEX-SDK is completely open source and lives in a GitHub repository[6]. The tools are hosted directly through GitHub Pages[7]. The realtime database used for implementation is Firebase[8]. When a programmer opens the tool in a new browser, they are automatically authenticated on Firebase with an anonymous user id, allowing them to already create content, render a robot, and interact with the robot. For continuity across devices, programmers can log in with a Google account. FLEX-SDK is currently tested only with a Chrome browser. The robot screen renderings are tuned assuming proportions of a Nexus 7 tablet.

5.3 Documentation

The GitHub page of FLEX-SDK has a wiki[9] describing the system and guiding the programmer through the use of the various tools. It is populated with instructional materials to allow a new programmer to create robot programs completely on their own. This includes tutorials for (1) starting and testing a robot, (2) remotely controlling a robot, (3) running existing programs on a robot, (3) configuring a robot and managing content, (4) creating new robot faces, (5) creating new robot I/O screens, and (6) writing new programs. In addition, the wiki involves documentation to help experienced programmers to contribute to FLEX-SDK software, to fix bugs and implement new features.

[6]FLEX-SDK code: https://github.com/mayacakmak/emarsoftware
[7]FLEX-SDK tools: https://mayacakmak.github.io/emarsoftware/
[8]Firebase: https://firebase.google.com/
[9]FLEX-SDK wiki: https://github.com/mayacakmak/emarsoftware/wiki

Table 1: A summary of the nine case studies in which FLEX-SDK was used for research and outreach.

	Study name & citation	Year	Study type	Programmers	Users	Tools used
CS-1	Disclosure [7]	2018	Controlled experiment	Undergrad researchers (N=2)	Participants (N=36)	FaceEdit, WoZ
CS-2	Robot vs. XR [8]	2019	Controlled experiment	Undergrad/grad researchers (N=3)	Participants (N=66)	FaceEdit, IOEdit, WoZ
CS-3	Teen operators [9]	2019	Observational	Participants (N= 20)	*same participants*	FaceEdit, WoZ
CS-4	Isolation companion [4]	2020	Participatory design	Participants (N=16)	*same participants*	FaceEdit
CS-5	Intervention design [5]	2020	Participatory design	Undergrad researchers (N=3)	Participants (N=30)	FaceEdit, IOEdit, programming
CS-6	Robot vs. workbook [5]	2021	Controlled experiment & observational	Undergrad researchers (N=2)	Participants (N=19)	FaceEdit, IOEdit, programming
CS-7	Embodiment kit [3]	2021	Participatory design	Undergrad researchers (N=2)	Participants (N=41)	FaceEdit, IOEdit, WoZ
CS-8	CS Women outreach, learning user-centered design	2019	Rapid prototyping	Participants (N=8)	*same participants*	FaceEdit, IOEdit, WoZ
CS-9	High-school outreach, learning basic programming	2019	Rapid prototyping	Participants (N=8)	*same participants*	FaceEdit, IOEdit, programming

6 CASE STUDIES

In this section we demonstrate how FLEX-SDK has been used as part of real human robot interaction studies and in outreach activities across nine case studies. Table 1 summarizes these case studies indicating which subset of FLEX-SDK tools were used in the study, the type of study in which the tools were used (e.g., participatory design session, observational study, controlled experiment, outreach), and who the *programmers* using FLEX-SDK tools and the *users* interacting with the created robots were. In the following we describe each case study in more detail and present observations about how FLEX-SDK was used in each of them.

Case Study 1. The first study involved designing a simple social robot (see Figure 9(a)) and using it to investigate how a user responds to different amounts and types of disclosure by the robot (see Figure 9(b)). The robot was operated through the WoZ interface populated with preset utterances (provided through the FLEX-SDK set up tool) to avoid typing during the study. The programmers who used FLEX-SDK were two undergraduate researchers (one from Psychology and other from Informatics) working on the project over the summer. The participants who interacted with the robot were recruited from a college population (N=36).

Case Study 2. The second case study involved another controlled experiment comparing the simple interactions with a mental health support robot with a virtual version of the robot on a screen or in Virtual Reality (VR). The physical robot was designed with FLEX-SDK content creation tools and controlled through the WoZ tool during the experiment.

Case Study 3. The third case study involved co-design sessions with teens to investigate how a robot listener should behave while a teen user shares their stressors with it. Teen participants acted as both the robot operator (see Figure 9(e)) and the user (see Figure 9(d)) to enact the dialog between the robot and the user. The operators used the WoZ tool to choose from a set of preset utterances or type their custom response. Participants who had extra time at the end also used the FaceEdit tool to design a face for the robot listener that would be welcoming and attentive (see Figure 9(f)).

Case Study 4. The next case study involved teen participants during the COVID-19 lockdown working to design a simple social robot companion in their home. Participants received a small tripod and used simple fabrication materials to create a robot embodiment around it. They used the FaceEdit tool to design the face of their robot. A variety of robots they created is shown in Figure 10.

Case Study 5. The fifth study involved participatory design of mental health support robots based on established evidence-based interventions, including Dialectic Behavioral Therapy (DBT) and Acceptance-Commitment Therapy (ACT). Three undergraduate researchers used tools in FLEX-SDK to create robot interactions that involve exercises from DBT and ACT. They iteratively refined the robot design and the interactions with feedback from teenagers over Zoom after they interacted with the virtual robot. Examples of virtual robot screens for different activities are shown in Figure 13.

Case Study 6. The next case study was a controlled experiment comparing the effect of a selected list of activities designed in CS-5, with worksheet-based activities that are currently used in practice. Participants had access to 10 different activities on a virtual robot during the course of a week and could freely interact with it.

Case Study 7. The seventh case study focused on the embodiment design for an actuated 4 Degrees-of-Freedom (DoF) robot core with two tablets. Participants from target user groups across three application areas (education, healthcare, community engagement) used

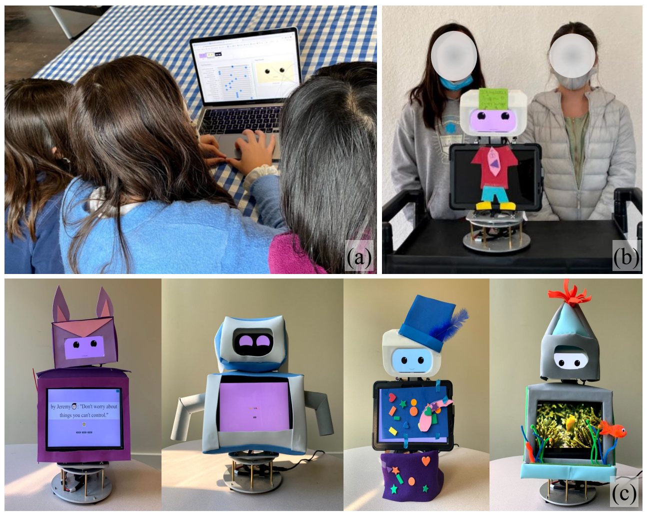


Figure 11: (a) Three children using the FaceEdit tool in FLEX-SDK to design a robot that would help them learn English as part of CS-7 and (b) two others showing their design. (c) Robot embodiment examples designed for different applications in CS-7 where participants used the FaceEdit and IOEdit tools to create the face and belly screens of their robots.

fabrication materials to design the robot embodiment and used the FaceEdit tool in FLEX-SDK to create the robot face that completes the robot's look. Figure 11(a) shows children who are English learners using the FaceEdit tool, and Figure 11(b) shows an example robot created by a pair of participants. Other robots created in this study are shown in Figure 11(c). In addition to designing robot faces, and in some cases I/O screens, the WoZ tool in FLEX-SDK was used as part of this case study to control the robot's motors so participants could see the robot animated and test its range of motion.

Case Study 8. This outreach activity was a two-day workshop designed for Women with disabilities who are studying CS and was intended to introduce them to an area of research. Our activity focused on design research and introduced the participants to methods in the human-centered design process, such as storyboarding, body-storming, rapid prototyping, and observational user studies. Participants in the workshop worked on designing robots that support mental health (similar to CS-6 an CS-7). They used FLEX-SDK to design the face and I/O screens for their robot (see Figure 12(a)) and created robot embodiment using prototyping materials. Then they used the WoZ tool to test the interactions that they designed with other workshop participants and other novice users from the workshop organization team (see Figure 12(b-c)).

Case Study 9. Our second outreach workshop was similar to CS-8, but instead intended for high-school students with disabilities and lasted a week. The participants designed a robot to support mental health and tested their prototypes with other workshop participants (see Figure 12(d-e)). The additional time of the workshop also allowed them to use the programming tool in FLEX-SDK as

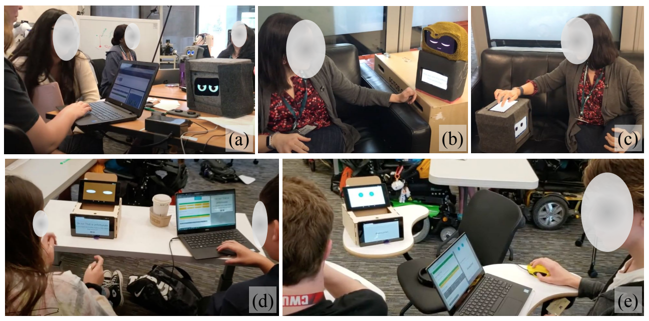


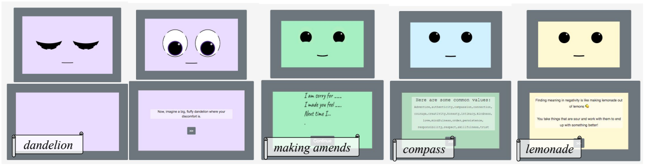
Figure 12: Case studies using FLEX-SDK as part of outreach activities: (a) Students setting up their robot in CS-8: a two day workshop for Women with disabilities in CS; (b) and (c) show a novice user interacting with the robots designed by students in CS-8; (d) and (e) show students testing robots that they created in CS-9: a week-long workshop for high-school students with disabilities.

they started learning basic programming constructs like loops and conditionals.

6.1 Observations

We make the following observations across the case studies:

- ***Enabling HRI research.*** Our case studies demonstrate that researchers with various backgrounds (including non-technical areas) can use FLEX-SDK to create WoZ or autonomous user studies to investigate HRI questions.
- ***An SDK for Rapid prototyping.*** Beyond their use by researchers to create interactions, FLEX-SDK tools like FaceEdit, IOEdit, and WoZ were used directly by participants to rapidly prototype robot looks or behaviors as part of participatory design studies or outreach activities.
- ***Face editor for all.*** Participants in our case studies including children, teens, and adults, people with various disabilities, and people with various backgrounds were able to use the FaceEdit tool without any instruction. They were able to tinker with different control elements in this tool and see the impact on the rendered face. They were able to explore the design space of faces to find the combination of parameters the expressed the face that they had in mind. They produced a wide range of robot faces that are very different from one another and from the faces of existing robot platforms (see Figures 10, 13, 11, 12).
- ***Levels of engagement.*** Across the different case studies we also observed that users engaged at different levels with operating and programming the robot: Level 1 – only using the WoZ to control the robot with existing content; Level 2 – creating new content for the robot (new faces or I/O screens); Level 3 – programming. This was not intentional in the design of the system, but rather emerged from the different use cases and the user experience levels.
- ***Simple programs.*** The use of the programming tool has so far been limited (CS-5, CS-6, CS-9) with most programs not taking full advantage of the available API and the expressivity of JavaScript. The example program given in Fig


Now, imagine a big, fluffy dandelion where your discomfort is.

I am sorry for ___
I made you feel ___
Next time I...

Here are some common values:
Adventure,authenticity,compassion,connection,
courage,creativity,honesty,intimacy,kindness,
love,mindfulness,order,persistence,
responsibility,respect,skillfulness,trust

Finding meaning in negativity is like making lemonade out of lemons 🍋

You take things that are sour and work with them to end up with something better!

Figure 13: Screenshots of virtual robots created with FLEX-SDK as part of CS-5 where participants co-designed robot-based interventions for mental health, and CS-6 where the interventions were compared to the current way those activities are performed with worksheets.

6 is substantially more complex than any of the programs created by researchers and participants in our case studies. For example, most programs developed in CS-5 (see Figure 13) involved the user walking through a series of screens and associated faces in sequence. The programs for these activities were simple scripts with a series of setScreen and setFace commands, with few places where the user input caused program branching through conditional statements or loops.

7 DISCUSSION

7.1 Contributions

Our work contributes a new open-source software development kit for designing and programming social robots. Although the system achieves known functionality leveraging known techniques, this functionality is not readily available to HRI researchers at the moment. There are currently no alternative system that allow turning two tablets in to a social robot within a matter of minutes. Most existing systems comparable to FLEX-SDK were developed for a particular robot system rather than an open-source, customizable hardware. On the other hand, starting from scratch to program two tablets to behave like a social robot, like the ones in our case studies, would take weeks or months of work and would only be possible with extensive programming expertise. Hence, we believe that the system contribution of our work will be valuable to the HRI community.

7.2 Potential Impact

The diversity of case studies presented in this paper demonstrates the types of research that FLEX-SDK would enable within the HRI community. By removing the need to purchase an expensive social robot platform and to install and learn complex robot programming software, FLEX-SDK can significantly lower the barrier to entry for HRI research and diversify our community. While the research enabled by FLEX-SDK has been largely focused on mental health, the same methods and procedures could be used in a wide range of applications of social robots. By giving full flexibility of robot embodiments and ability to customize the robot look and behavior, FLEX-SDK can allow the discovery of new use cases for social robots and unexplored creative form factors (see [3] for examples).

7.3 Limitations

The key limitation of FLEX-SDK, in terms of the types of autonomous social robot programs that can be created with it, is in the use of perceptual capabilities. Even the simplest robot that consists of a single head tablet has a camera available, but FLEX-SDK currently does not provide any API functions to take advantage of camera perception. Since many HRI applications can benefit from such perception (e.g., face tracking and recognition) and given the growing availability of browser-based image processing capabilities, extensions of the API to include more perception is a high-priority for future work. Many other extensions of FLEX-SDK are possible. We expect the new features and tools in FLEX-SDK will be driven by new use cases as it has been in the past three years and we hope that more members of our community can contribute to its development.

7.4 Lessons Learned

FLEX-SDK has been under development for more than five years and has gone through many revisions based on our experience using it in our research. Next we discuss some of the lessons learned in this process to inform future versions of the system as well as the development of other similar systems.

What did users of FLEX-SDK have trouble with?

Although we observed that participants created a wide range of robot faces using the FaceEdit tool (Section 6.1), they did not take advantage of its full expressiveness. In the space of possible robot faces, many participants' faces seemed to stay close to the initial, default face settings (e.g., Figure 10). In contrast, we recreated faces of commercial robots, originally created by professional designers with expert animation tools, spanning a much wider range of the space (Figure 4).

Similarly, although our the programming tool gives full expressivity of a general-purpose programming language (JS), programs created across our case studies were very simple (Section 6.1). This seemed to be a result of our system's way of separating *content creation* from *programming*. Programmers spent a lot of time creating different robot faces and I/O screens, but their programs resulted in being simply sequentially going through the screens in order. Although this pattern was not adopted intentionally in the beginning, it was reinforced by the case studies. As a result of our participants' emphasis on content creation, we implemented more features in content creation tools rather than expand the API, in order to achieve a desired new functionality (e.g., ability to include emojis on the screen).

What did users do that was different from what was expected?
The WoZ system served an unexpected purpose for programmers in learning about the system's API. Programmers who were not introduced to the WoZ tool would learn about the different functions in the API by writing single-line programs and running them to observe the effect on the robot, and explore different parameter settings. On the other hand, programmers who had used the WoZ extensively before starting to program already had an intuitive grasp of the API. This observation points towards a broader role that WoZ systems can have in learning to program robots. Controlling the robot through WoZ allows people to quickly get exposed to the full API, observe the robot in action, and get a sense

of the impact of different parameters. Programmers typically learn these while simultaneously trying to program. We hope to run a controlled study in the future to better characterize the impact of using WoZ in learning to program a robot.

What did users want that the system did not have? Our system was designed to work on a browser to maximize accessibility and reduce barriers to entry, and it used cloud functionality to have different parts of the system to communicate. While programmers appreciated the workflow enabled by this design, end-users consistently stated that they would not want a robot that is connected to the cloud for privacy and security reasons.

Robotics researchers who have used FLEX-SDK have been particularly interested in extending the API with new actions and inputs to allow for more autonomous behaviors. In extending the SDK with new *actions* one insight from our studies is that we need to clearly distinguish between (i) instantaneous actions like 'setFace()' and (ii) temporally extended actions like 'playSound()' and allow a way to catch the ending of such extended actions. Without this, we ended up having programs where the programmers had to estimate the duration of a sound and add sleep() statements to their programs.

In extending the SDK with new *inputs* we expect the key challenge will be making the programmer aware of possible values of the input–this was already an issue with inputs such as 'waitForButton()' where the user themselves named the buttons. It will become a bigger challenge with camera or microphone input that can return continuous values or infinitely many discrete values, or can return with an error. As with actions, maintaining the distinction between waitFor_ and get_ statements is also important for maintaining the intuitiveness with such input extensions.

ACKNOWLEDGMENTS

We are grateful to all the participants that were part of our studies. This work was supported by the National Science Foundation, awards IIS-1924435 "NRI: INT: Program Verification and Synthesis for Collaborative Robots" and SES-1734100 "NRI: INT: Design and Development of a Social Robot for Gathering Ecological Momentary Stress Data from Teens."

REFERENCES

[1] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. 2021. A Survey on End-User Robot Programming. *arXiv preprint arXiv:2105.01757* (2021).

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

[3] Patrícia Alves-Oliveira, Matthew Bavier, Samrudha Malandkar, Ryan Eldridge, Julie Sayigh, Elin A Björling, and Maya Cakmak. 2022. FLEXI: A Robust and Flexible Social Robot Embodiment Kit. In *Designing Interactive Systems Conference*. 1177–1191.

[4] Patrícia Alves-Oliveira, Elin Björling, Patriya Wiesmann, Dwikat Heba, Simran Bhatia, Kai Mihata, and Maya Cakmak. 2022. Robots for Connection: A Co-Design Study with Adolescents. In *(IEEE International Conference on Robot and Human Interactive Communication (Ro-Man))*.

[5] Patrícia Alves-Oliveira, Tanya Budhiraja, Samuel So, Raida Karim, Elin A Björling, and Maya Cakmak. 2022. Robot-Mediated Interventions for Youth Mental Health. *Design for Health* (2022).

[6] Patrícia Alves-Oliveira, Alaina Orr, Elin A Björling, and Maya Cakmak. 2022. Connecting the Dots of Social Robot Design From Interviews With Robot Creators. *Frontiers in Robotics and AI* (2022), 127.

[7] Elin A Björling, Honson Ling, Simran Bhatia, and Kimberly Dziubinski. 2020. The Experience and Effect of Adolescent to Robot Stress Disclosure: A Mixed-Methods Exploration. In *International Conference on Social Robotics*. Springer, 604–615.

[8] Elin A Björling, Honson Ling, Simran Bhatia, and Jeff Matarrese. 2021. Sharing stressors with a social robot prototype: What embodiment do adolescents prefer? *International Journal of Child-Computer Interaction* 28 (2021), 100252.

[9] Elin A Björling, Kyle Thomas, Emma J Rose, and Maya Cakmak. 2020. Exploring teens as robot operators, users and witnesses in the wild. *Frontiers in Robotics and AI* 7 (2020), 5.

[10] Michael Jae-Yoon Chung, Justin Huang, Leila Takayama, Tessa Lau, and Maya Cakmak. 2016. Iterative design of a system for programming socially interactive service robots. In *International Conference on Social Robotics*. Springer, 919–929.

[11] 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 *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2352–2357.

[12] Sarah Gillet, Ronald Cumbal, André Pereira, José Lopes, Olov Engwall, and Iolanda Leite. 2021. Robot Gaze Can Mediate Participation Imbalance in Groups with Different Skill Levels. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*. 303–311.

[13] 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.

[14] Dylan F Glas, Takayuki Kanda, and Hiroshi Ishiguro. 2016. Human-Robot Interaction Design Using Interaction Composer: Eight Years of Lessons Learned. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. IEEE Press, 303–310.

[15] Kelleher R Guerin, Colin Lea, Chris Paxton, and Gregory D Hager. 2015. A framework for end-user instruction of a robot assistant for manufacturing. In *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 6167–6174.

[16] Justin Huang and Maya Cakmak. 2016. Programming by Demonstration with User-Specified Perceptual Landmarks. *arXiv preprint arXiv:1612.00565* (2016).

[17] Justin Huang, Tessa Lau, and Maya Cakmak. 2016. Design and evaluation of a rapid programming system for service robots. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 295–302.

[18] Alisa Kalegina, Grace Schroeder, Aidan Allchin, Keara Berlin, and Maya Cakmak. 2018. Characterizing the design space of rendered robot faces. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. 96–104.

[19] Tino Lourens. 2004. TiViPE-Tino's visual programming environment. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. IEEE, 10–15.

[20] Tino Lourens and Emilia Barakova. 2011. User-friendly robot environment for creation of social scenarios. In *International Work-Conference on the Interplay between Natural and Artificial Computation*. Springer, 212–221.

[21] Hai Nguyen, Matei Ciocarlie, Kaijen Hsiao, and Charles C. Kemp. 2013. ROS Commander (ROSCo): Behavior creation for home robots. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 467–474.

[22] 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.

[23] David Porfirio, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2018. Authoring and verifying human-robot interactions. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 75–86.

[24] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 46–51.

[25] Finn Rietz, Alexander Sutherland, Suna Bensch, Stefan Wermter, and Thomas Hellström. 2021. WoZ4U: An Open-Source Wizard-of-Oz Interface for Easy, Efficient and Robust HRI Experiments. *Frontiers in Robotics and AI* 8 (2021).

[26] Allison Sauppé and Bilge Mutlu. 2014. Design patterns for exploring and prototyping human-robot interactions. In *Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1439–1448.

[27] Katie Winkle, Emmanuel Senft, and Séverin Lemaignan. 2021. LEADOR: A Method for End-to-End Participatory Design of Autonomous Social Robots. *arXiv preprint arXiv:2105.01910* (2021).