Enabling End-Users to Create Real-World Robot Applications through Visual Programming Interfaces and Automation

Michael Jae-Yoon Chung

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Maya Cakmak, Chair

Rajesh P.N. Rao, Chair

Dieter Fox

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Enabling End-Users to Create Real-World Robot Applications through Visual Programming Interfaces and Automation

Michael Jae-Yoon Chung

Co-Chairs of the Supervisory Committee:
Professor Maya Cakmak
Computer Science and Engineering

Professor Rajesh P.N. Rao
Computer Science and Engineering

Interactive robots in human environments have the potential to change our daily lives by providing education, eldercare, entertainment services, and more. However, creating new applications and behaviors for such robots is complex for many reasons, including the challenges of prototyping and testing. This thesis starts by presenting our research on designing and evaluating an information-gathering application for a mobile robot in a university venue and exploring the use of robots for gathering customer feedback in the hospitality industry. Our goal is to enable non-experts, such as salespeople and designers, to create and explore new robot applications without requiring additional technical expertise. To that end, we share our experiences of iteratively adapting a robot programming system for creating socially interactive robots in commercial spaces. Based on these experiences, we identified the ability to *express concurrency* and *modify low-level behaviors with ease* as two important requirements for interactive robot programming systems. We categorize common concurrency patterns and present a framework for identifying programming constructs required to express concurrency; we then implemented and investigated three representative concurrency interfaces for a block-based visual robot programming system and assessed them with a systematic evaluation and a user study. To modify low-level behaviors, we present an

iterative robot program repair approach that enables programmers or the robot's users to easily change program details that govern a robot's low-level behaviors to improve interaction quality; we evaluated our approach with both simulation and human experiments. Throughout the document, we discuss our findings, results, insights, and the limitations of our approaches. We conclude the thesis by envisioning directions for future work.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First, I would like to thank my advisors Profs. Maya Cakmak and Rajesh P.N. Rao for their persistent guidance and support over the years; I was blessed to have them on my Ph.D. journey. I am also grateful to Prof. Dieter Fox for creating an inspiring robotics research environment and Dr. Blake Hannaford for reviewing my thesis. Between 2016 and 2018, I had the opportunity to work and conduct research at a robotics startup company, Savioke; I greatly appreciate the generous support and invaluable experience I gained there. Justin Huang showed me that finishing a doctorate program was possible; Christian Fritz showed me ways to apply research ideas in practice; Kathleen Tuite and Maxwell Forbes showed me how to live a creative life; Seungyeop Han showed me the limitlessness of being a graduate student; Sandy Kaplan showed me how to write like an academic; and Franklin Pajarito made me feel safe on nights I worked late in the lab. They made my graduate school years unforgettable. I am deeply thankful to other peers, mentors, mentees, and staff members at the Computer Science & Engineering department for shaping who I am today. Last but not least, I pay tribute to my wife, Ka Young Kim, and my parents, Ki-Young Chung and Sung Ran Lee, for their endless support and love. Without them, this thesis would not have been possible.

Chapter 1

INTRODUCTION

Over the last decade, we have witnessed the increasing emergence of robots in human environments. Startup companies like Savioke, Aethon, Bossa Nova Robotics, and Diligent Robotics have deployed autonomous indoor robots for delivery and inventory monitoring in commercial spaces, such as hotels, retail stores, and hospitals [228]. Social robots are also becoming more popular, for example, personal robots like Jibo have become available to consumers [97], and companies like Catalina Health and No Isolation have begun to provide robot-based services for chronic care management and education, respectively [129, 8]. Most robots deployed in human environments are built for a *specific purpose* and have a limited set of behaviors or *contents*. We believe the critical next step for unlocking the true potential of these robots is democratizing the process of creating robot applications and behaviors.

Creating new applications for robots operating in human environments is challenging. Because ubiquitous robotics is a relatively young technology, existing product design techniques are not directly applicable to robot application design [155]. The physicality of robots makes application prototyping time consuming and costly, and the autonomous nature of robot systems when combined with unpredictable humans makes testing difficult. Robot applications often have multiple stakeholders, such as robot service companies (e.g., a delivery service robot company), the company that uses the robot (a hotel), the robot's user (a guest receiving an item), and bystanders (passerby guests), which also makes robot application design unique. Social robots exhibit a variety of behaviors. For example, a storytelling robot should have as many stories as there are books for an e-reader like Amazon Kindle. One common approach to enlarging variety is to let users create robot behaviors via a programming system. Designing a robot programming system requires choosing from a large

design space of programming systems and evaluating expressivity vs ease-of-use trade-offs. In fact, creating robust and natural interactive behaviors is demanding even for professional programmers.

Our research goal is to facilitate the creation of new robot applications and behaviors using human-centered design techniques and programming systems tailored to using interactive robots in human environments.

To that end, we investigated the following approaches:

1. *Human-Centered Design.* ISO standards define human-centered design this way. *"Human-centred design is an approach to interactive systems development that aims to make systems usable and useful by focusing on the users, their needs and requirements, and by applying human factors/ergonomics, and usability knowledge and techniques."* In the spirit of human-centered design [49], our research begins by focusing on users. For example, we conducted *need-finding interviews* to identify the underlying needs of users (Chapter 4, Chapter 5) and *user surveys* to better understand the users' requirements (Chapter 3, Chapter 4). To identify the needs and requirements of multiple stakeholders of the robot application (e.g., those who employ the robot and the robot's users), we gathered data from all identified stakeholders (Chapter 4).

2. *Real-World Deployment.* Our research involves several real-world deployments to study robot applications in their actual context of use. We used the robot deployment not only to check the feasibility of proposed solutions but to identify technical, interaction, and design challenges. For example, we conducted Wizard-of-Oz and autonomous system deployment experiments in target environments such as an office building and hotels, respectively (Chapter 3, Chapter 4). We also conducted a sequence of deployment studies to identify technical and design challenges and adapt proposed applications using a robot programming system *iCustomPrograms*, iteratively (Chapter 5). Our experiences from the deployment experiments were used to identify key problems with using robot programming systems in real-world settings, viz., concurrency issues

and the difficulties of adjusting low-level behaviors, which we further investigated in Chapter 6 and Chapter 7.

3. *Block-Based Visual Programming.* We investigated the use of block-based visual programming tools to create robot applications deployed to commercial spaces (Chapter 5) and studied concurrency issues within them (Chapter 6). While other approaches can help non-experts program robots, we chose a block-based visual programming tool due to its pervasiveness and the research of Huang, who demonstrated the feasibility of this tool for helping non-expert users program mobile manipulation programs [91].

4. *Robot Program Repair with Implicit Feedback.* Program repair is a formal method for automatically fixing faults in programs given specifications [98, 150]. We introduce an approach to improve interactive behaviors of robots by repairing robot programs that represent robot behaviors as finite state machines. Our approach derives repair objectives from corrective feedback on the interaction provided by the robot's user, which we call *implicit feedback.* Our approach is iterative: it updates the repair objective as more implicit feedback becomes available, and it uses Bayesian inference to solve the repair problem. The main benefit of our approach is it makes the tuning of robot program parameters, crucial for creating robust interactions, easier for both lay and professional robot programmers.

1.1 Robot Platforms

Our research made use of three robot platforms: DUB-E, the Savioke Relay, and TaRo. Photographs of each robot are shown in Fig. 1.1.

1.1.1 DUB-E

DUB-E is a custom-built mobile robot for robotics research. It is based on the MetraLabs Scitos G5 mobile base [140] expanded with a structure providing support for sensors and

Figure 1.1: Robot platforms used in our research. From left to right, DUB-E, the Savioke Relay, and TaRo.

user interface devices. Specifically, DUB-E is equipped with an Allied Vision Manta G609 camera on a tilt unit for data collection, two Asus Xtion Pro depth cameras (one forward and one backward facing), a Hokuyo UTM-30LX laser range finder for navigation, and an LCD display and speakers for communicating its intent.

1.1.2 Relay

The Savioke Relay is an autonomous mobile robot for providing guest room delivery service [173]. The robot is approximately 3 feet tall and weighs 100 pounds, has a lockable interior bin, and displays a touchscreen that is mounted facing forward. The robot stays in its docking station and charges its battery when not in use. Our research used the Relay robot for designing and evaluating new use cases in indoor spaces.

1.1.3 TaRo

TaRo (**Ta**blet **Ro**bot) is a custom-built, idealized social robot for human-robot interaction research. Taro consists of a face, a GeeekPi 7 inch touchscreen for displaying messages and buttons, and a neck, a 5-DoF Open MANIPULATOR-X robot arm for making head gestures

such as nod and shake. Taro is capable of speaking, recognizing speech, and detecting faces by processing data obtained via a speaker, a microphone, and a webcam attached to the connected computer.

1.2 Thesis Overview

1.2.1 Thesis Statement

Creating new applications and behaviors for interactive robots in human environments is facilitated by (1) human-centered design techniques adapted to robotics, and (2) robot programming systems that are usable by non-experts.

1.2.2 Summary of Contributions

1. We present an end-to-end information-gathering framework, findings from conducting formative studies regarding user and technical requirements, an implemented system, and its evaluation in a university venue to demonstrate feasibility (Chapter 3).

2. We present a series of studies exploring the viability of a custom feedback-collection application for mobile robots in hotels and share our design recommendations based on study results, some of which were conducted in real hotels (Chapter 4).

3. We report a series of findings regarding technical requirements and design implications based on use of a non-expert-friendly programming system for creating socially interactive robot applications in commercial spaces, such as an airport and hotels (Chapter 5).

4. We present a framework for creating concurrency-friendly, visual block-based, interactive robot programming interfaces and the three representative interfaces we created using the framework. We also share our insights regarding the ease-of-use of the interfaces based on a systematic evaluation and a user study involving non-roboticist programmers (Chapter 6).

5. We present an approach that enables non-roboticist programmers to create interactive robot programs and iteratively improve them by having them interact with the robot's users. We present experimental results demonstrating the feasibility of this approach (Chapter 7).

Chapter 2
RELATED WORK

In this chapter, we describe the three areas of robotics research that are closely related to our research.

2.1 Human-Centered Design for Service Robots

Our work is part of the research being done in the field of human-robot interaction (HRI), which focuses on designing and evaluating real-world robot applications. Over the last decade, HRI researchers have employed product design methods such as *need finding* and *participatory design* to understand users' needs and related constraints and requirements for domains of interests, such as home organization [156] and guiding visitors in tourist sites [106], airports [100] and office buildings [11]. Findings from these studies have been abstracted into themes [100, 11], frameworks [156], and visual maps of user experiences [106] and used to derive design recommendations. Some of this work considered multiple stakeholders – e.g., the target service industry workers who will be working with robots and the visitors who will be served by robots [100] – while other work considered only main target users, e.g., family members involved in organizing their houses or blind pedestrians who need help navigating in buildings [156, 11]. While such research presents insightful recommendations, we believe it is also critical to evaluate a potential application early in the deployment process and in real-world settings (for example, via prototyping) to learn critical design and implementation lessons as soon as possible (Chapter 3, Chapter 4).

A body of HRI work focuses on evaluating robotic systems in the wild [33, 69, 101, 23, 104, 80]. Researchers in this domain ask questions concerning the feasibility and performance of the proposed systems, e.g., "Can the robotic system successfully perform the given

task?" and "How well does it work?" This research attempts to understand why proposed systems work (or do not work) and to identify factors influencing robot usage. For example, researchers verified the feasibility of deploying fully autonomous robots as guides in public places [33, 23, 104] and as walking group assistants in a care site [80]. Researchers reported the robustness of robotic systems as being a key requirement for using robots in real-world settings [104, 80]. They also reported potential benefits, such as enhancing the motivation of older adults [80], and identified potential and identified technical challenges, like managing engagement [101, 23], which sparked follow-up research efforts (e.g., [103, 5]). Compared to these studies, our work puts more emphasis on understanding the requirements and opinions of multiple stakeholders of our proposed use case before developing it (Chapter 3, Chapter 4).

Our robot application design work (Chapter 3 and Chapter 4) is most closely related to work that applies a human-centered design approach to service robot application development. Three prominent examples are Snackbot [115], a delivery robot in an office building; TOOMAS [51], a shopping guide robot in a home improvement store; and Sacarino [166], a bellboy robot in a hotel. The research teams for these projects employed design processes that involved multiple phases of prototyping with potential users in target environments to get continuous feedback during development. Recently, researchers proposed adapting a *lean UX* design approach from the startup field for identifying a commercial social robot application and applied the proposed method for rapidly prototyping an assistant robot in Sydney International Airport [212]. Some work presents a system or framework that supports iterative robot application design and demonstrates it by developing robot applications in real-world settings [124, 93, 41]. Like this work, our research also emphasizes the importance of robot users and the context of the robot application and evaluates prototypes in real-world settings. However, our research focuses on two less well explored mobile robot applications, viz., and information gathering application in offices and guest feedback collection in hotels (Chapter 3, Chapter 4).

2.2 End-User Robot Programming

Research on end-user robot programming aims to produce intuitive abstractions of robot programming languages and develop usable user interfaces for programming. Researchers developed several visual programming languages based on abstractions like flowcharts [169, 67, 3], state machines [198], behavior trees [162], block-based imperative languages [93, 41, 92], and trigger-action rules [119]. Systems have been developed for programming mobile or tabletop manipulators [3, 93, 92, 161, 198, 65], humanoid robots [67, 68, 119], and social robots [31]. Others investigated alternative modes of interaction such as natural language [72, 31], bodystorming [167], and tangible interfaces [188, 65]. Like previous work in end-user robot programming, we aim to enable non-roboticists to author robot behaviors. Our research focuses on specific problems inherent in end-user robot programming, such as concurrency issues (Chapter 6) and parameter tuning problem of interactive robot programming (Chapter 7), which we discovered from our experiences of using a robot programming system in the wild (Chapter 5).

2.3 Robot Programming with Concurrency

Concurrency is essential for programming interactive robot behaviors. A subset of prior work presents ways to express concurrency in a robot programming system. Lourens and Barakova's textual language [127] and Chorepographe, a default programming system for the Nao humanoid robot [169], provide generic operators such as wait, parallel, and sequence for coordinating actions and sensor inputs. Interaction Composer enables the handling of asynchronous events using the interrupt module, which executes the robot action defined within the module when a monitoring event is triggered [67, 68]. Robot programming systems have taken a human-centered approach to provide specialized ways to express frequently desired concurrent actions [41, 48]. Some robot control systems that aim to make authoring socially intelligent, reactive and complex robot behaviors easy are based on a mathematical model or programming paradigm that emphasizes concurrency support, such as timed Petri

net [36], behavior tree [136], and the reactive programming paradigm [17]. Although this past work provides specific ways for users to express concurrency, its priority is developing a robot programming system. Our research further investigates and evaluates different ways of supporting programming interactive robots with concurrency (Chapter 6).

2.4 Applications of Formal Methods in Robot Programming

The robotics community has long been interested in applying formal methods, such as verification and synthesis, to robotics problems. Techniques in formal verification have been applied to assess the correctness of concurrent and time-critical programs [52] for checking both general and application-specific properties [142]. Researchers also used program synthesis for finding a plan or a controller in the context of navigation [54], mobile manipulation [147], or the multi-robot planning problem [216] that satisfies the specifications often expressed in a temporal logic language. Other work explored alternative specification languages, such as structured natural language [111], or adaptation of an existing robot program in a new environment interactively with a human user, e.g., in the context of robot soccer [86] or tabletop manipulation [26]. Most recently, Hammond et al. introduced a system that can automatically recover from errors while running end-user-created, service mobile robot programs [76]. The iterative repair approach we investigated (Chapter 7) is most closely related to the work of Holtz et al. on automatically repairing robot-soccer playing programs, which are represented as FSMs, using sparse state corrections from a programmer with a MaxSMT solver [86, 87]. Our approach, however, focuses on repairing social robot programs with feedback provided by the human who is interacting with the robot.

Recently, the applications of formal methods in robotics was extended to include scenarios relevant to human-robot interaction. Formal verification has been used to ensure effective and reliable human-robot teamwork for astronaut-robot collaboration in space missions [25] and validate the safety requirements of robotic assistants [226]. Program synthesis has been used to create safe human-in-the-loop controllers from high-level temporal specifications [121] and programs for neuro-rehabilitation with a social robot [113]. Porfirio et al. [168] contributed a

system for authoring human-robot interaction programs for social robots and verifying that the program meets social norms or best practices discovered by the human-robot interaction research community. The same researchers also developed a system to synthesize interactive programs from traces of two people acting out the interaction scenario [167]. While most work in this category focuses on reasoning about high-level human-robot interaction task structure, the recent work of Kshirsagar et al. [112] involves synthesizing low-level controllers for human-robot handover tasks. Similarly, the iterative repair approach we investigate involves program repair at a lower level, maintaining a high-level structure of programs (Chapter 7).

Chapter 3

DESIGN AND EVALUATION OF INFORMATION GATHERING SERVICE FOR INDOOR MOBILE ROBOTS

3.1 Background

Many day-to-day tasks in large human environments (office buildings, hospitals, retail stores, or warehouses) require up-to-date knowledge about the environment. However, human environments are inherently dynamic due to human activity. People constantly move around and change the state of the environment through their interactions within it. The more populated the environment, the more difficult it is to predict its state at a given time. Since many human tasks depend on accurate knowledge of the state of the environment, a large part of our daily tasks involve gathering information. This chapter explores the design and evaluation of a service that lets users outsource information gathering tasks. We provide this service with autonomous mobile robots.

The use of robots for information gathering is not a new idea. Outdoor mobile robots – including ground, aerial, and underwater robots – have been used for search and rescue missions [96] and space and oceanic frontier exploration [215, 12]. However, the use of indoor mobile robots for similar purposes has remained unexplored.[1] We call indoor information gathering robots *InfoBots*. InfoBots can respond to questions of building occupants (e.g. ,"Is my advisor in her office?") or requests (e.g. "Let me know when my advisor arrives in her office."). They can also monitor the state of the building and report unusual events (e.g., open office door at 2 am) or maintenance needs (e.g., broken lightbulbs). One way to gather information without physically going to the locality of interest is to use a telepresence robot [201]. However, this solution does not provide strong productivity benefits since it requires

[1]This statement was true at the time of [38] (2014).

the user to actively control the robot. A different solution includes stationary cameras installed in the environment [148]; however, this requires modifications to the environment, cannot provide full coverage of the environment with high-density information, and presents greater privacy and security concerns than information-gathering robots. Moreover, InfoBots can conduct other tasks, like fetch-and-delivery, simultaneously with information gathering tasks.

In this chapter, we present (1) a formative study involving a user survey and a Wizard-of-Oz deployment experiment designed to gather requirements for mobile-robot-based information gathering, and (2) a framework for providing an information-checking service, a type of information gathering. Using the framework, we implemented a system that takes a natural language question (e.g., "Is Mike Chung in the robotics lab?") and returns a picture that best captures the requested information. We then evaluated the system based on information collected from a formative study of system users and on information collected from DUB-E, a mobile robot deployed in a university building.

3.2 Information Gathering in Human Environments

First, we discuss different types of information gathering in human environments and describes steps for *information checking*, a type of information gathering that we focus on in this chapter.

3.2.1 Information Gathering Task Types

We categorize indoor information gathering tasks into four types:

- *Checking* involves going to a particular location and reporting specific information about the current state of the world.
- *Searching* involves going to multiple candidate locations until a specific type of information is captured, and then reporting the location.

- *Monitoring* involves going to a particular location and waiting until a particular state change information is captured, and then reporting the time of occurrence of the event.
- *Summarizing* involves passively gathering information at an arbitrary location or while traveling along different locations for other tasks, and then providing an cumulative report of salient information (states or state change events).

Among the four task types that cover wide-range of services that InfoBots could provide, our research focuses on information *checking*. Questions suitable for the checking tasks are primarily constrained by the locality of the answer and the robot's ability to capture the answer within its sensory horizon.

3.2.2 Information Checking

We operationalize information checking as answering questions about the environment (Fig. 3.1). We represent the environment (e.g. a building) as a collection of *locations* (e.g. cse100, cse101, ...) at some predefined level of abstraction (e.g. room). The locations have *location attributes* (e.g. John's office, seminar room 101) and *variable states* (e.g. person present/not present, room occupied/not occupied). The location attributes can be used to identify a corresponding location. The variable states are dynamic and their value at any given time is unknown to remote users; therefore, questions are assumed to be about the variable states of locations.

The input to the system is a user question and the output is an answer or a sensory recording that contains an answer. First, the location and the variable state that best describe the given question are identified by a set of location attributes. The identified location and variable state specify a target map pose and a target sensor configuration required to capture the requested information. The robot navigates to the target pose, configures sensors to the target configuration, and takes a sensor recording (e.g. an image). Finally, the system returns the sensor recording or inferred answer from the sensor recording to the user.

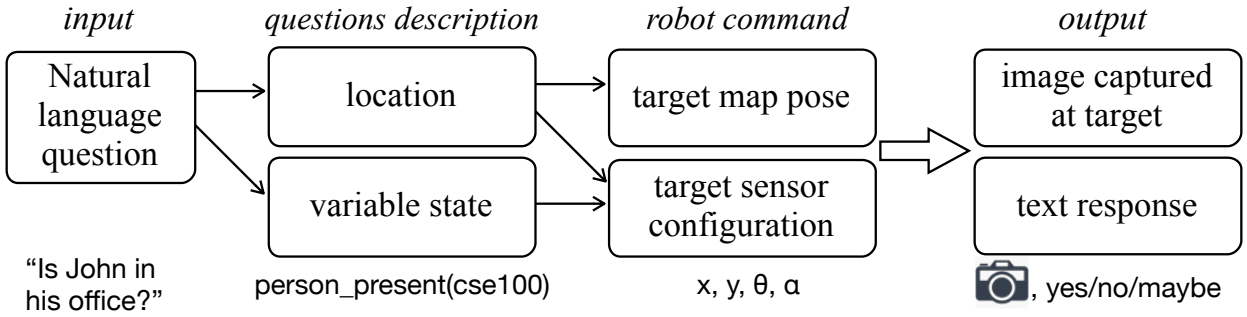


Figure 3.1: Steps for information checking. The solid arrows represent the input processing steps and the hollow arrow represent the step involving physical robot actions (e.g. navigation) that produces the outputs.

3.3 Requirements for Information Checking

The primary stakeholders of InfoBots providing information checking are the users requesting the information. Meeting their requirements is critical for the actual adoption of InfoBots. Hence, our requirement gathering efforts are focused on those users. Nonetheless, the views of the bystanders, particularly those about whom information might be requested, are also important. We come back to considerations about these secondary stakeholders in (Sec. 3.9).

3.3.1 Survey Design

We created a survey with two aims: (i) determining the types of information that would be most useful and (ii) identifying constraints and requirements for how information would be requested and provided. To inform this survey with a list of questions that people might be interested in asking InfoBots, we conducted semi-structured interviews with 8 participants (4 M, 4 F, ages 24–70) from the Computer Science & Engineering building at our institution. The following information checking requests were most commonly mentioned in these interviews:

- *Is John in his office?*
- *How many people are in the lounge?*

- *Are there any empty tables in the study room?*
- *Are there any bagels at the coffeeshop?*
- *Is there free food in the kitchen?*
- *Is the conference room occupied?*

After introducing the idea of InfoBots, our survey presents these six information requests and asks the following multiple-choice questions regarding usefulness, usage frequency, and time constraints for each request:

- The ability to ask this type of question would be [5: Very useful, 1: Useless]
- I would ask this type of question [5: Multiple times a day, 4: Every day, 3: Once/twice a week, 2: Once/twice a month, 1: Never]
- I would require a response [5: Immediately, 4: Faster than human, 3: Same speed as human, 2: About half the time as human, 1: No rush]

The help text for each question indicates that the given question is just an example and that their response should address the category of questions similar to the particular example. In addition, an open-ended question asks for instances of other similar requests the users might have.

3.3.2 Findings

We administered our survey to occupants of two buildings at the University of Washington: Computer Science & Engineering (CS) and Law School (Law). We scouted both buildings before instantiating the example questions to make sure the questions are meaningful and accurate. We advertised the survey through mailing lists targeting undergraduate and graduate students, faculty, and staff. We received 80 responses from the CS (23 M, 25 F, 32 unspecified, age range 18–65 with mean 29.46 and standard deviation 11.64) and 31 responses from the Law (10 M, 11 F, 10 anonymous, age range 29–68 with mean 48.26 and standard deviation 12.42).

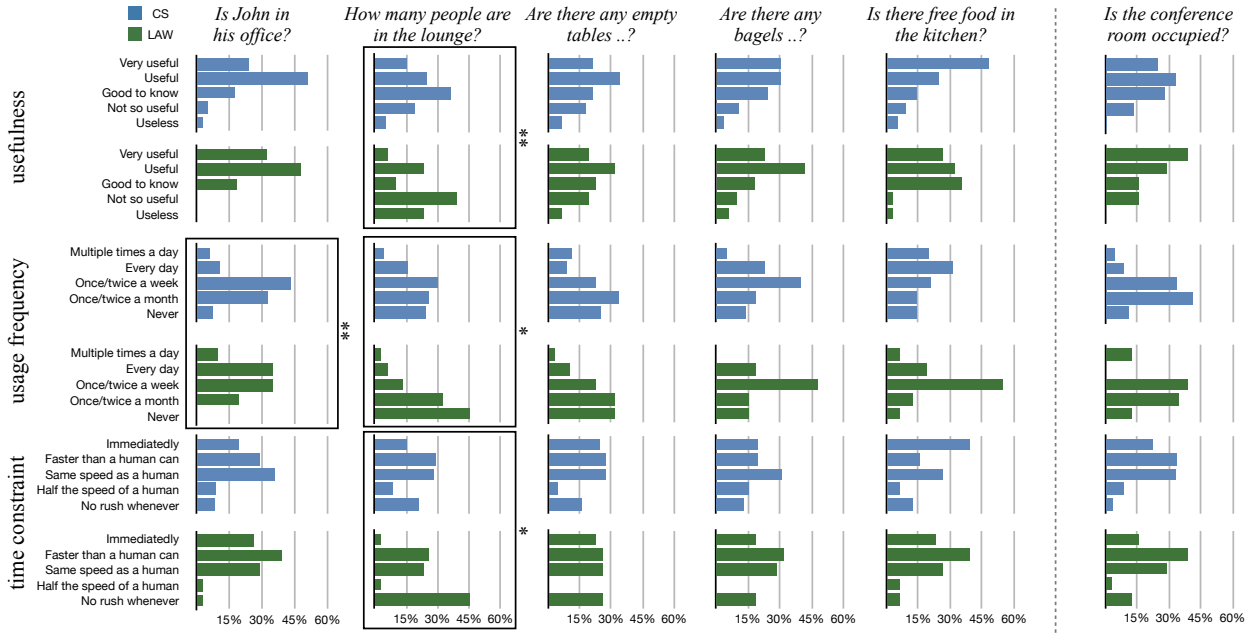


Figure 3.2: Multiple choice survey results. The results are organized by example information check question (column), survey question type (row), and population (color). Each plot represents the response distribution of the corresponding survey question. Pairs of CS and Law results that are significantly different are marked with boxes; (*) and (**) denote $p < .05$ and $p < .01$, respectively.

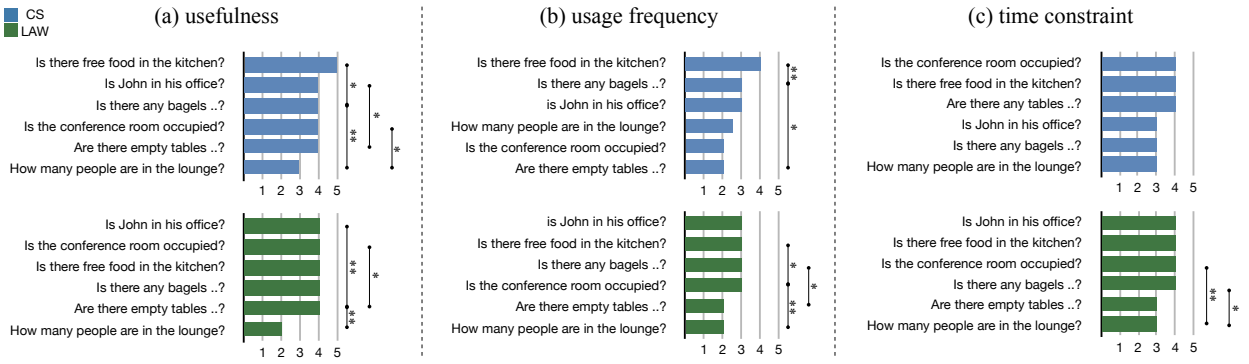


Figure 3.3: Responses to sample information check questions sorted by median. The response distribution mean was used to break ties. (*) and (**) denote $p < .05$ and $p < .01$, respectively. The results are organized by survey question type (column), and population (color).

Fig. 3.2 summarizes the responses from the multiple choice part of the survey. To compare the differences between response distributions from CS and Law, we ran Wilcoxon-Mann-Whitney tests on 18 CS and Law response distribution pairs (three question types × six sample questions). We found four significantly different pairs (shown in Fig. 3.2). An analysis of differences among responses to different information request types is presented in Fig. 3.3. We make the following observations.

Usefulness. The results indicate that people think InfoBots can be useful. On average 84% of CS respondents and 79% of Law respondents thought that the answer to a state check question would be "Good to know" or better ("Useful," "Very useful"). Among the six questions, the one for checking the presence of a person and the one asking about the availability of food placed in the top three for both CS and Law (Fig. 3.3a). The question that was rated as "Useless" the most times was the one asking about the number of people in a room.

Usage frequency. The usage frequency results were lower in scale than the usefulness results. Respondents considered the questions useful even though they anticipated asking them infrequently. For example, more people indicated that they would "Never" ask a question (CS: 16%, Law: 19%) than they indicated the question as being "Useless" (CS: 4%, Law: 6%). This is a positive finding as the robot serving a whole community of residents would not be able to handle high frequency requests from individual users.

Time constraints. Respondents had high expectations in terms of the response speed. 49% of CS respondents and 52% of Law respondents wanted a response "Immediately" or "Faster than a human can". 30% of CS respondents and 27% of Law respondents wanted a response at the "Same speed as a human". These requirements are currently unrealistic even if an InfoBot were serving a single user. Nonetheless, this shows that (i) there are some questions that people are okay with getting a response to in human-speed or slower, and (ii) some people are willing to wait longer for certain questions.

Variance across buildings. Responses from the two buildings were not significantly different for most questions. While this suggests that the role of InfoBots might not vary too much

across different buildings, culture and workflow of different buildings might impact usage of InfoBots. For instance, our survey indicated that the ability to check whether someone is in their office would be used significantly more frequently in the Law building than the CS building.

3.4 Wizard-of-Oz Deployment Experiment

Our survey indicated that InfoBots might provide a useful service; however, what people say does not always match what they actually do. This is particularly true for robotic services since most people do not have experience with them[155]. To study the practical usage of InfoBots, we deployed a semi-supervised, Wizard-of-Oz (WoZ) controlled InfoBot in the computer science building where the survey was conducted.

3.4.1 Robot and Web Interface

Our InfoBot is DUB-E, a custom-built mobile robot for research. The front-end of our system is a web interface organized as a feed of questions posted by users (Fig. 3.4). To collect unconstrained natural questions, we let users post free-form questions using the text field. When a user submits a question, they could choose whether a question should be visible to all users ("public" mode), whether notification emails should be sent as status updates, and a timeout for when the answer would no longer be needed. Once a question is submitted, it is placed in the user's private feed with the service status, as well as in the public feed if the question was in the public mode. Public questions have a "comments" panel and a "Thank you robot" button.

The robot and the web interface are connected by a back-end system supervised by a human operator. This system monitors user activities and alerts the operator when a question is posted. The supervision interface allows the operator to accept or reject a questions, move the mobile base, play sound files, display messages on the LCD display, and post a response to the question. It also visualizes the data from the on-board sensors (e.g. images from a camera).

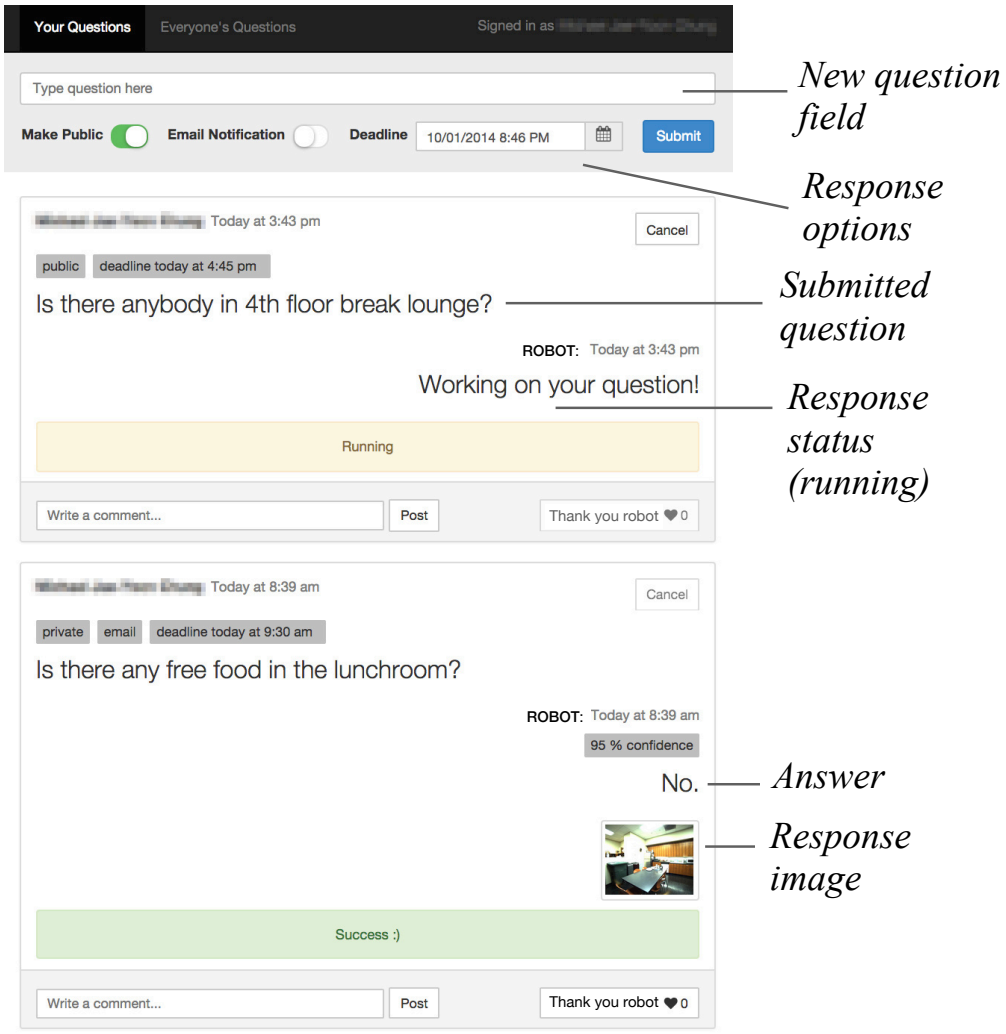


Figure 3.4: InfoBot front-end web interface.

3.4.2 Experiment Procedure

Our system was deployed for four business days (9am to 5pm). We recruited users by emailing graduate student, undergraduate student, staff, and faculty mailing lists. In the recruitment email, we provided a brief introduction of the information gathering service and the service website address. In addition to the recruitment email, we displayed posters containing a picture of the robot and the service website address on multiple bulletin boards

in the building. To prevent non-residents from signing up for the service, the system only allowed people with a valid computer science email account to sign up.

When a user asked a question on the website, the operator received a question and decided to accept or reject the question. The operator was instructed to only accept the checking type questions that could be answered by analyzing a static image from the robot visiting a location in the building. After the operator made an acceptance decision, they supervised the robot to go to the target location in the building, position the camera, and save an image from the camera. Subsequently, they answered the question by only looking at the acquired image, and then updated the web interface with the answer and image. To communicate the robot's intent to bystanders, the operator displayed the message, "Working on behalf of {username} on question {question}" on the LCD display while the robot was navigating. Additionally, when the robot arrived at its destination, it verbalized the message through the on-board speaker using text-to-speech with a male voice.

At the end of the forth day, we sent a short questionnaire to users who asked at least one question. This questionnaire asked participants about their experience with the InfoBot service, their satisfaction with the service, and their likelihood of using the InfoBot when it becomes permanently installed in the building. In addition, the questionnaire asked for open-ended feedback.

3.4.3 Findings

Question Types

Over the course of the four days when the service was available, a total of 88 questions were posted. The "deployment" column in Table 3.1 summarizes distributions of these questions. During the experiment, the operators categorized the questions into two groups, checking and non-checking (search, monitor and summarization questions or non-questions, Sec. 3.2.1). 80% of the total questions were of the checking type, which the operator accepted, and the other 20% (non-checking questions) were rejected. In the latter case, the operator posted

Table 3.1: Question Types

Information Gathering Task Types	**deployment**	**post-survey**
checking	70 (80%)	18 (78%)
non-checking	18 (20%)	5 (22%)
total	88	23

Checking Task Types ($\kappa = 0.89$)	**deployment**	**post-survey**
presence	53 (76%)	10 (56%)
state	17 (24%)	8 (46%)
total	70	18

Presence Targets ($\kappa = 0.93$)	**deployment**	**post-survey**
person	32 (60%)	4 (40%)
food	11 (21%)	5 (50%)
mail	4 (6%)	1 (10%)
other	7 (13%)	0
total	54	10

"This question is not for me" as a response.

Questions in the checking category were further grouped into two categories: *presence* (76%) and *state* (24%). Two authors coded 70 checking questions and measured inter-coder agreement using Cohen's κ (0.89). The most common presence type checking questions were "Is there anyone in {location}?" and "Is {person} in his/her office?" (Table 3.2, 1–2). We further categorized presence by its target objects; two authors coded 53 questions and measured inter-coder agreement using Cohen's κ (0.93). Although there were more than 20 target object categories, only three objects (person, food, and mail) appeared more than once. We merged the other target object categories into a new category called "other." Out of all presence questions about objects, users asked most about the presence of food and mail, e.g. "Is there any food in the downstairs kitchen?" and "Is there anything in my mailbox?" (Table 3.2, 3–4). For the state type checking questions, we observed a wide variety of questions ranging from checks about the accessibility of services (e.g. "Is the door to the conference room open?" and "Is the reception still open?") to noise conditions

Table 3.2: Sampled Questions from Deployment Experiment

	questions	**information gathering task type**	**checking task type**	**targets**
1.	"Is there anyone in {location}?"	checking	presence	person
2.	"Is {person} in his/her office?"	checking	presence	person
3.	"Is there any food in the downstairs kitchen?"	checking	presence	food
4.	"Is there anything in my mailbox?"	checking	presence	mail
5.	"Is the door to the conference room open?"	checking	state	N/A
6.	"Is the reception still open?"	checking	state	N/A
7.	"How noisy is it in the atrium right now?"	checking	state	N/A
8.	"Is it raining outside?"	checking	state	N/A
9.	"Is there an empty conference room in the Computer Science building?"	other	N/A	N/A
10.	"Has {person} arrived yet today in the CS building?"	other	N/A	N/A
11.	"Which meeting room has the best visibility of the {landmark} today?"	other	N/A	N/A
12.	"What do you look like?"	other	N/A	N/A
13.	"Are there any mirrors in the building?"	other	N/A	N/A
14.	"Who let the dogs out? :)"	other	N/A	N/A

Table 3.3: Demographics

Job Titles	**# of users**	**# of questions**	**avg. questions per user ± std.**
faculty/staff	16	33	2.06 ± 0.93
graduate	20	44	2.20 ± 2.04
undergraduate	9	11	1.22 ± 0.67
total	45	88	1.95 ± 1.5

(e.g. "How noisy is it in the atrium right now?") or weather conditions (e.g. "Is it raining outside?") (Table 3.2, 5–8).

The non-checking category included both search (Table 3.2, 9–10) and monitoring (Table 3.2, 11) type questions. Several questions in the *other* category were clearly submitted with the purpose of challenging the system (e.g. "What do you look like?" and "Are there any mirrors in the building?") or simply as jokes (e.g. "Who let the dogs out?") (Table 3.2, 12–14). Such requests were rejected by the operator.

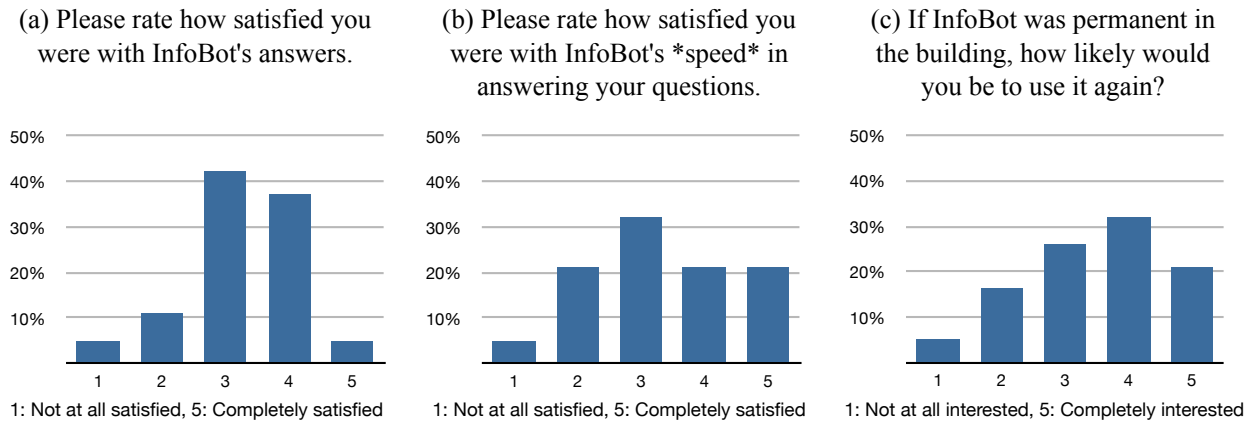


Figure 3.5: Post survey results.

Demographics

Table 3.3 presents the general statistics of the service usage. We identified three groups of users based on their occupation: faculty/staff, graduate students, and undergraduate students. Although the number of graduate students or faculty/staff is much smaller than that of undergraduate students, graduate students and faculty/staff used the service more often. This can be explained by the fact that users in these groups spend more time in the building and therefore were more likely to perform everyday tasks in the building.

Post-Deployment Survey

Among 45 users who asked at least one question on the website, 20 users participated in the post survey. 12 respondents reported the InfoBot actually answered their questions, and 7 respondents reported the answer returned by the InfoBot was actually useful. Two respondents reported that the InfoBot was not answering their questions because it replied with "This question is not for me." This happened because they were not asking the checking type questions. Two respondents mentioned that while the InfoBot's text answer did not answer their question, they could extract the answer from the associated image.

We asked about the users' satisfaction with the Infobot's answers and its response speed

(Fig. 3.5). In terms of satisfaction with the answer, the majority (84%) were "Neutral" or better ("Satisfied", or "Completely satisfied"). In terms of satisfaction with the response speed, only 5% of the users were "Not at all satisfied" despite their initial high expectation (Sec. 3.3.2). More than half of respondents (53%) were interested ("Interested" or "Definitely interested") in using the system if the InfoBot became permanently deployed (Fig. 3.5c). In the last part of the survey, we asked what questions users would ask if the InfoBot were permanent. To compare questions listed in response to this survey and questions asked during the deployment, we categorized them in the same manner as we did with the deployment questions. The result is shown in Table 3.1 "post-survey" column. We observed that around 20% of the new questions were still non-checking questions. Among the new checking questions, the percentage of the *state* checks was almost doubled (from 24% to 46%) as compared to state check questions that were actually asked during the deployment.

3.5 Overview of the Information Checking Framework

The results of the formative studies gave us insights into the types of questions people might ask if an autonomous framework was to be developed (Sec. 3.4.3). Moreover, the intuition that images can be a powerful medium when conveying the answers to questions was supported by two users indicating that, while the text answer did not answer their question, they could extract the answer from the associated image (Sec. 3.4.3).

In this section, we present a framework to autonomously respond to questions asked by users. Fig. 3.6 shows a system we implemented using the framework.

3.5.1 Problem Description

The goal is to find the best viewpoint v^* in which the requested information $\mathcal{I} \in \{0, 1\}$ is present, given a natural language question s. We assume the robot is operating in a dynamic environment described by a map M, and we have access to the database D containing domain

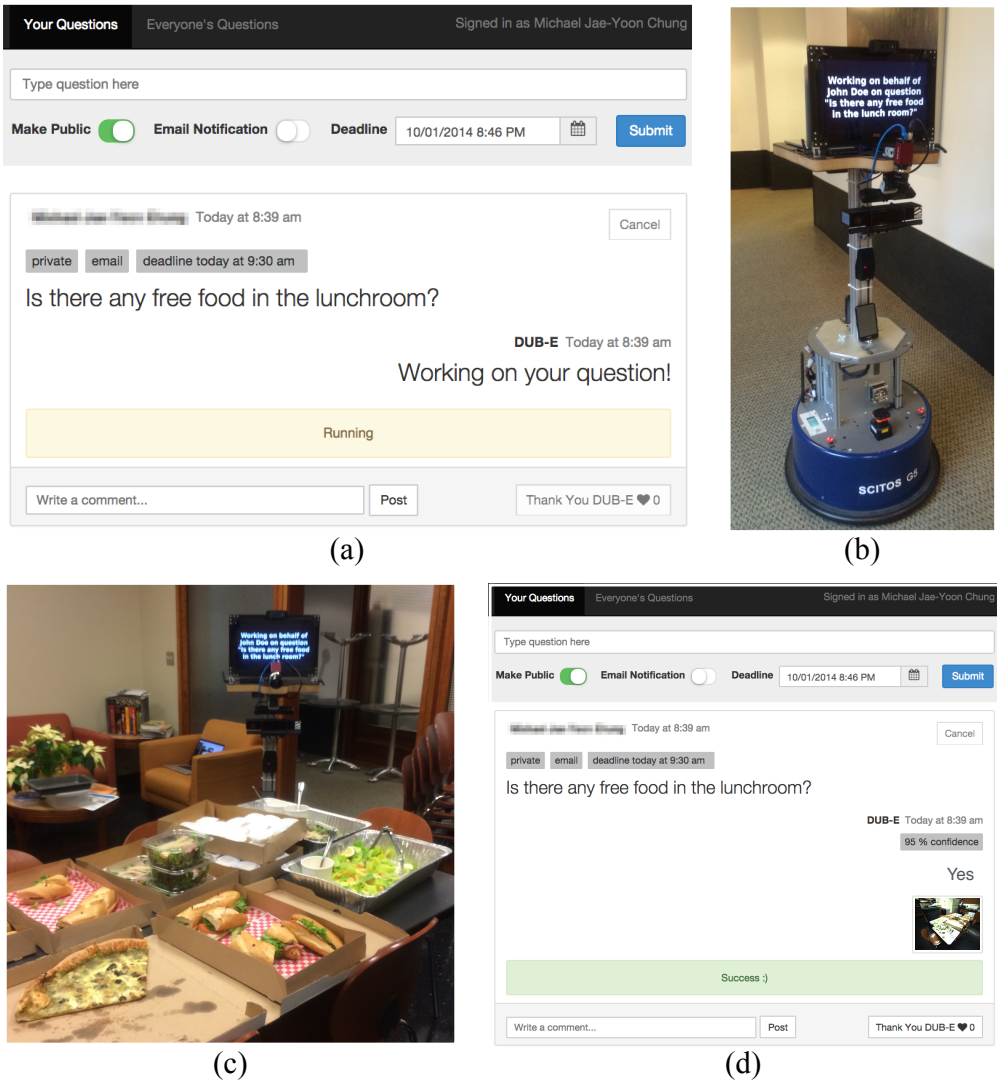


Figure 3.6: The end-to-end system: (a) a question is submitted through the web interface; (b) robot estimates the initial configuration, navigates to the destination while iteratively refining the estimate; (c) it captures the image containing the requested information; (d) the image is delivered to the user through the web interface.

knowledge about M. We formulate our problem as

$$v^* = \underset{v}{\text{argmax}} P(\mathcal{I} = 1 | v, s; M, D) \tag{3.1}$$

where

$$P(\mathcal{I}=1|v,s;M,D)=\sum_{z}P(\mathcal{I}=1|v,z;M,D)P(z|s;D) \tag{3.2}$$

$$\approx\max_{z}P(\mathcal{I}=1|v,z;M,D)P(z|s;D) \tag{3.3}$$

where z is a descriptor of the requested information. Factoring the problem in this way allows us to independently estimate the optimal **viewpoint** given a concrete descriptor with $P(\mathcal{I}=1|v,z;M,D)$ and the **natural language parse** of the question as a descriptor with $P(z|s;D)$.

We define $P(z|s;D)$ as a distribution over information descriptors z for each question sentence s. For example, given the sentence $s=$ "Is there anyone in the robotics lab?" and the record in D that identifies "Mike Chung" as $person$ and "robotics lab" as $cse101$, the desired information descriptor $z=\texttt{presence}(person,cse101)$ is a tuple precisely describing the requested information in s.

The distribution $P(\mathcal{I}=1|v,z;M,D)$ for estimating viewpoints can be decomposed as follows (M,D omitted to keep notation uncluttered):

$$P(\mathcal{I}=1|v,z)=\sum_{x}P(\mathcal{I}=1|x,z)P(x|v) \tag{3.4}$$

where x are locations on the map (e.g. cells in a 3D occupancy map), $P(\mathcal{I}=1|x,z)$ models the presence of the information at location x and $P(x|v)$ models the visibility of location x from viewpoint v.

3.6 Parsing Natural Language Questions

The deployment experiment in Sec. 3.4 revealed that there are two types of checking questions: (a) the questions concerned with *presence* of things at a location, and (b) the questions concerned with *state* of a location. Reflecting this observation, the information descriptor z takes one of two forms: $\texttt{presence}(l,t)$ or $\texttt{state}(l)$, where l is a room in the building (e.g.

$cse100$), and t is a target type (e.g. $person$). More formally, we define the information descriptor as a tuple $z = (\tau, l, t)$ where $\tau \in \{\texttt{presence}, \texttt{state}\}$, $l \in \{cse100, cse101, ...\}$, and $t \in \{person, object, N/A\}$.

Parsing a language input question s to an information descriptor z is equivalent to evaluating $P(z|s; D)$. We first process s by using Stanford CoreNLP Natural Language Parsing Toolkit [134] to extract part-of-speech (POS) tags, a context-free phrase structure tree, and results from applying co-reference resolution. We merge all outputs from the CoreNLP to a parse tree s' by copying the output parse tree and replacing its leaf nodes with the input words and the POS tag pairs, and then use the results from the co-reference resolution to replace the subtrees corresponding to the referring words with the subtree corresponding to the referred words. For example, given s = "Is Mike Chung in his office?", the sentence extracted from s' is "Is Mike Chung in Mike Chung's office?".

Given s' we evaluate:

$$P((\tau, l, t)|s'; D) = \alpha \max_i \mathbf{1}(T_i^\tau(s')) \times \left\{ \max_j d(L(T_i^\tau(s')), A_j(l)) + \max_k d(G_i^\tau(s'), B_k(t)) \right\}. \quad (3.5)$$

where

- α is a normalization constant,

- $T_i^\tau(s')$ is an ith τ type relation template that can detect words describing a location and a target type in s'. Templates use relationships between tags (e.g. check if a node has children with tags PP and NP) and predefined keywords (e.g. check if a word paired with a IN POS tag equals to the locational preposition such as "in", "at", etc.) to detect the words. $\mathbf{1}(T_i^\tau(s'))$ returns a boolean variable indicating whether $T_i^\tau(s')$ fit on s' or not.

- $L(\cdot)$ and $G(\cdot)$ operators return detected words describing a location and a target type,

respectively, from applying a template $T_i^{\tau}(s')$. Using the s' mentioned in the earlier example, $L(T_i^{\texttt{presence}}(s'))$ = "Mike Chung's office" and $G(T_i^{\texttt{presence}}(s'))$ = "Mike Chung" for some i.

- $A_i(l)$ returns ith words describing l and $B_j(t)$ returns jth words describing t by looking up the data stored in D. For example, $A_i(cse102)$ ="Mike's Office" and $B_j(person)$ ="Mike Chung" for some i, j.

- $d(\cdot, \cdot)$ function measures the similarity between two text inputs (e.g. Levenshtein distance).

- D is a database contains records of (i) room and its name pairs, and (ii) target type and its attributes (e.g. target instance name). The records in the database can be manually constructed or (as in our case) imported from the external database (e.g. the building database).

If the input sentence is not a checking type question, then the distribution $P((\tau, l, t)|s'; D)$ will not be proper; no relation templates $T_i^{\tau}(s')$ can cover the input s'.

In Sec. 3.5.1, we approximate the summation in Eq. 3.2 with the max in Eq. 3.3. In other words, we are only considering the most likely information descriptor instead of all possible information descriptors. In our domain, most of times, the ambiguity in z comes from ambiguous natural language descriptors of the target location in s. In such case, even if the InfoBot considers all possible choices of z to find v^*, it ends up heading to a single location. Therefore, propagating uncertainty further in this particular case does not help making better decision. One practical solution for choosing z is if the $P(z|s; D)$ is not peaked at a single z, the system can prompt a question asking for clarification to the user.

3.7 Viewpoint Estimation

As mentioned in Sec. 3.5.1, estimating the best viewpoint for answering the question asked by a user is equivalent to evaluating Eq. 3.4. In the following, we describe how we model

the environment M and the terms involved in Eq. 3.4.

3.7.1 Environment Model

Our environment M is a tuple (M^{2D}, M^{3D}, M^T).

- M^{2D} is a 2D occupancy grid map with a resolution of 0.05m in which each grid cell is identified by its Cartesian coordinates in a global coordinate frame and described as either empty, occupied, or unknown. M^{2D} is acquired by mapping using a method developed by Grisetti et al.[74] and post-processed to only contain static information (e.g. walls and stationary furniture). M^{2D} is mainly used for navigation and for annotations in the database.

- M^{3D} is a 3D occupancy grid map similar to M^{2D} with an additional 3rd (height) dimension with a resolution of 0.05m. M^{3D} provides a richer representation of the environment; however, in dynamic environments it can quickly become outdated. Hence, we continuously update it with incoming depth data using Hornung et al.'s method [88]. M^{3D} is used for reasoning about visibility.

- M^T is a topological map in which each topological node is a candidate *place* for the viewpoints that the robot can gather information from. For each place, there is a discrete sets of candidate *orientations* that specify the *viewpoint*. When computing Eq. 3.1, we search for v^* in a M^T to make our problem tractable.

While we use existing mapping algorithms for acquiring M^{2D} and M^{3D} [74, 88], we use a custom algorithm for generating M^T as described in the following.

Topological Mapping. The topological map allows the system to constrain the problem of viewpoint estimation to a discrete subset of all possible viewpoints. This makes the problem tractable, but also results in a commitment that could harm performance. Therefore, it is important to select a discretization that properly supports the problem at hand.

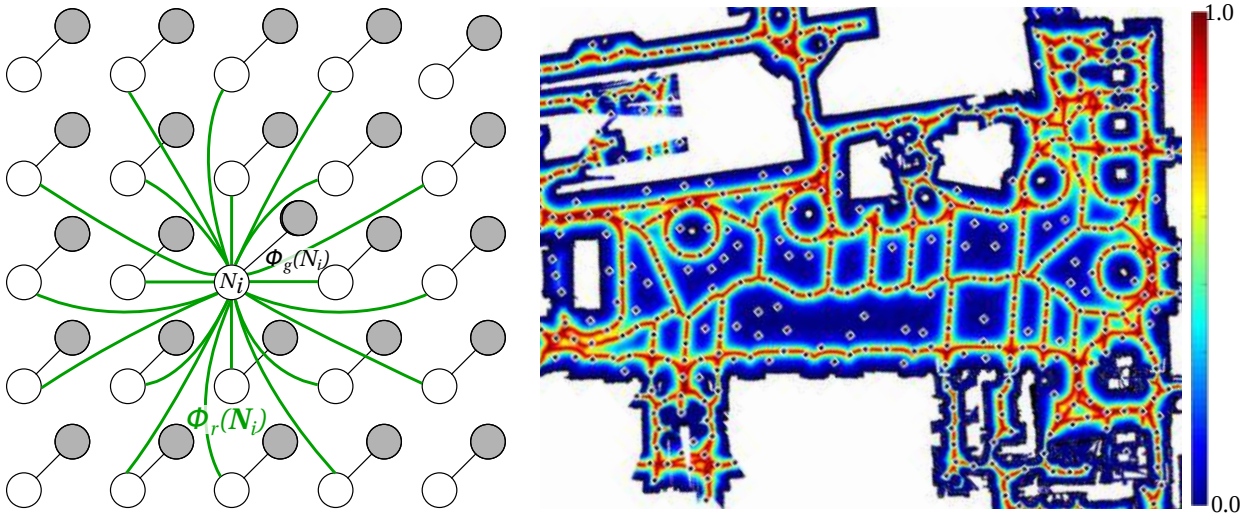


Figure 3.7: Topological mapping. (a) Probabilistic graphical model illustrating the distribution from which topological maps are sampled. (b) A typical example of a set of topological nodes on top of the values of $\phi_g(N_i)$ for each pixel of an occupancy grid map.

We generate topological maps from a probability distribution $P(M^T|M^{2D})$ that models the relevance of locations to the task and distributes topological *places* accordingly. The Markov Random Field illustrating the distribution is shown in Fig. 3.7a and corresponds to:

$$p(N|M^{2D}) = \frac{1}{Z}\prod_i \phi_r(N_i)\phi_g(N_i), \tag{3.6}$$

where $N_i \in \{0, 1\}$ determines whether a place exists at location i and $N_i = \{N_j : j \in \text{neighborhood}(i)\}$ for a local spatial neighborhood of 1m radius.

The potential function $\phi_g(N_i)$ models the relevance of a location for the task and is defined in terms of three potentials calculated from the 2D metric map:

$$\phi_g(N_i) = \phi_o(N_i)\left(\phi_c(N_i) + \phi_v(N_i) - \phi_c(N_i)\phi_v(N_i)\right), \tag{3.7}$$

where:

- ϕ_o depends on the distance d_o to the nearest obstacle and is calculated similarly to the cost map used for the navigation algorithm [135]. ϕ_o equals 0 for distance smaller than the radius r of the robot base and $exp(-\alpha(d_o - r))$ otherwise.

- $\phi_v = exp(-\beta|d_o - d_v|)$ depends on the relation between the distance d_o and the fixed distance d_v that provides good visibility of obstacles in the map.

- $\phi_c = exp(-\gamma d_c)$ depends on the distance d_c to the nearest node of a Voronoi graph of the 2D map. This promotes centrally located places since central locations are often safe for navigation.

Overall, the definition of $\phi_g(N_i)$ ensures that candidate viewpoint locations are located only in areas that will not lead to a collision with obstacles and are either preferred due to their central location or visibility properties. The potential $\phi_r(N_i)$ ensures that places are distributed within certain distance to one another, by enforcing low probability for locations that are close to other existing places.

We employ Gibbs sampling to perform the maximum a posteriori inference and choose samples corresponding to maps with highest posterior probability. A typical example of a generated set of topological nodes for a single floor of a building is shown in Fig. 3.7b. For each place, we assume a discrete set of *orientations* evenly spread across the full circle. The orientation and the metric position of a place fully specify a *viewpoint*. The resulting map M^T is expressed as a set of viewpoints M_i^T, with each view anchored in a metric map M^{2D}, i.e. $M_i^T = (M^{2D}, x, y, \theta)$.

3.7.2 Information Presence Term

As we want to reason about the information presence and its visibility with the most complex environment representation, we use M^{3D} for the space of x in $P(\mathcal{I} = 1|x, z; M, D)$ and

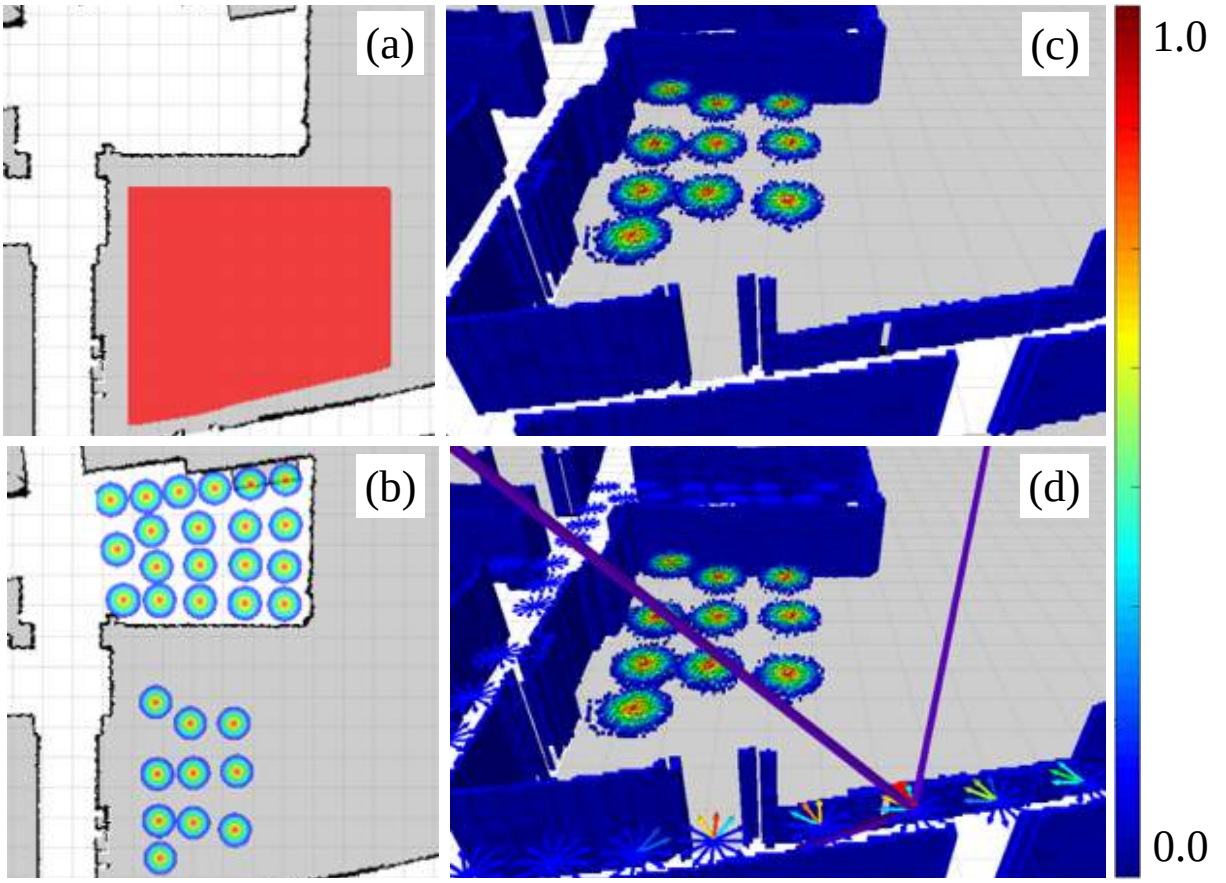


Figure 3.8: Annotations, information presence term, and viewpoints. (a) and (b) display respectively a room and person annotations overlaid on M^{2D}. (c) shows the information presence term computed using (a) and (b). (d) shows the evaluated viewpoints $P(v, \mathcal{I} = 1|z; M, D)$ as colored arrows and the camera field of view of the optimal viewpoint drawn with the purple lines.

$P(x|v; M^{3D})$[2] in Eq. 3.4. However, in dynamic environments where M^{3D} is continuously changing, defining $P(\mathcal{I} = 1|x, z; M^{3D})$ for all possible M^{3D} is impossible. Instead, we collect static *annotations* $P(\mathcal{I} = 1|y, z; M^{2D})$ where y is a cell in , then transform them to $P(\mathcal{I} = 1|x, z; M, D)$ with the most up-to-date M^{3D}. An example of $P(\mathcal{I} = 1|x, z; M, D)$ is shown in Fig. 3.8c.

[2]The visibility distribution $P(x|v; M, D)$ is dependent only on M^{3D}, therefore $P(x|v; M, D) = P(x|v; M^{3D})$.

The first term in Eq. 3.4, $P(\mathcal{I} = 1|y, z; M, D)$, models the presence of the information specified by z at location y (a cell in M^{2D}). It is computed based on annotations provided a priori and stored in the database D. Annotations are associated with polygon regions (e.g. the room annotation in Fig. 3.8a) or a set of discrete points (e.g. annotation of *person* in Fig. 3.8b) on M^{2D}. They are divided into two groups; one corresponding to a location name (l in z) and another corresponding to the presence of the target type in the specified region (t in z). The location name annotations are associated with polygon regions and have the same value $P(\mathcal{I} = 1|y, z; M^{2D}) = 1$ at all cells y within the specified region. On the other hand, the target type annotations are associated with a discrete set of points with non-zero probability where they are likely to be present (e.g., an object near table, a person near desks). Assuming independence between two groups, we have $P(\mathcal{I} = 1|y, z; M^{2D}) = P(\mathcal{I} = 1|y, l; M^{2D})P(\mathcal{I} = 1|y, t; M^{2D})$.

We transform $P(\mathcal{I} = 1|y, z; M^{2D})$ to $P(\mathcal{I} = 1|x, z; M)$ using a $2D$ to $3D$ coordinate conversion function $f|M^{3D} : Y \rightarrow X$. The function $f|M^{3D}$ maps an input 2D coordinate to a 3D coordinate by extending the input 2D coordinate with a height value. Assuming the objects of interest are usually located at a certain distance above the ground (e.g. people), we sample the height value from a Gaussian distribution with the mean μ and the standard deviation σ.

3.7.3 Visibility Term

The second term in Eq. 3.4, $P(x|v; M^{3D})$, models the visibility of cells x in M^{3D} from viewpoint v. We compute it using raytracing in M^{3D}. We set $P(x|v; M) = 0$ for (i) x that are not visible and (ii) x that are located farther than θm from the camera origin. We set $P(x|v; M) = 1.0$ for all x that are within the camera's cone of visibility. We experimented with other $P(x|v; M)$ such as $P(x|v; M)$ dependent linearly on the Euclidean distance between x and the camera origin; however, the performance differences were negligible in our experimental setting. Finally, while the robot is executing the task, it might discover certain v are not reachable. For those v, we set $P(x|v; M) = 0$ all x.

3.7.4 Iterative Refinement

Once v^* is computed based on Eq. 3.1, the robot starts navigating to v^*. Since the robot is operating in a dynamic environment, M in Eq. 3.1 might be outdated. As a result v^* may not provide the best viewpoint with respect to the new M. Assuming the availability of the component that can track the changes in M with incoming data, we address this problem by letting the system continuously re-evaluate $P(\mathcal{I} = 1|v, z; M, D)$ until there is no change in v^*. Note that we do not re-compute $P(z|s; D)$ in Eq. 3.2 since it is not dependent on M. Once the robot reaches the final v^*, it saves an image from the on-board camera (Fig. 3.6c) and returns this image as its response (Fig. 3.6d).

3.8 Experiments and Results

3.8.1 Natural Language Parsing

We evaluated our input question parsing component on the real user questions collected during the deployment experiment described in Sec. 3.4. The labels for the questions were acquired by a coding process performed by two of the authors. Labeling involved writing an information descriptor z^* for each question sentence s as $z^* = \text{argmax}_z P(z|s; D)$. For the locations l in information descriptors $z = (\tau, l, t)$ we used the 295 unique locations extracted from the building database.

In order to test our system's ability to correctly parse questions that involve information checking, we first ran a checking vs. non-checking classification experiment. Our system classifies a question as "checking" if the output distribution from the parser is proper (i.e. $\sum P(z|s, D) = 1$), and as "non-checking" otherwise. We attained an accuracy of 74%, a precision of 94% and a recall of 71% (# of true positives: 48, true negatives: 17, false positive: 3, false negatives: 20). A high precision rate is desirable from the robot's perspective as false positives will result in executing the wrong task. An example of a false positive is "What does Mike Chung look like when he's not at his desk?" (negation "not"). Examples of false negatives include "How many LEDs are on the wall in the Atrium?" (two locational PP

phases), "Is the service elevator in the CSE building operational?" (implicit/unknown target location), and "What color is Hank wearing today?" (implicit/unknown target location).

For the 65 questions that were correctly identified as checking questions, we evaluated our system's ability to extract the corresponding information descriptor. Our parser achieved an accuracy of 95% in classifying the question subtype τ, 89% in classifying the location l, 82% in classifying the target type t, and 72% in correctly classifying the full information descriptor $z = (\tau, l, t)$.

3.8.2 Viewpoint Estimation

We evaluated our viewpoint estimation component with two experiments involving the real robot in the computer science department building.

Experimental Setup. We used the custom-built mobile robot based on the MetraLabs Scitos G5 mobile base expanded with a structure providing support for sensors and user interfaces (Fig. 3.6b and Fig. 3.6c). A high-resolution Allied Vision Manta G609 camera with 97° horizontal and 79° vertical view angle, which is used for providing images to the users, is attached to the robot at 1.31m above the ground. An Asus Xtion Pro depth camera is placed at 1.25m above the ground to collect depth images for the purpose of building 3D maps. Another backward facing Xtion depth camera and a Hokuyo UTM-30LX laser range finder are also placed on-board for navigation purposes.

The 2D occupancy maps M^{2D} used in our experiments were collected prior to running experiments. The initial 3D occupancy maps M^{3D} were constructed from the corresponding 2D occupancy maps by extending occupied cells to the default wall height (2m) or default window height (0.85m) depending on their location. For topological map M^T generation, we used the following parameter values: $\alpha = 5$, $\beta = 8$, $\gamma = 10$. The discrete orientations of the viewpoints were distributed in the topological map every 30°. For all maps, we only represented the open spaces such as corridors and breakout areas to avoid going into building occupants' offices during working hours.

The 295 location annotations were imported from the building database and the person

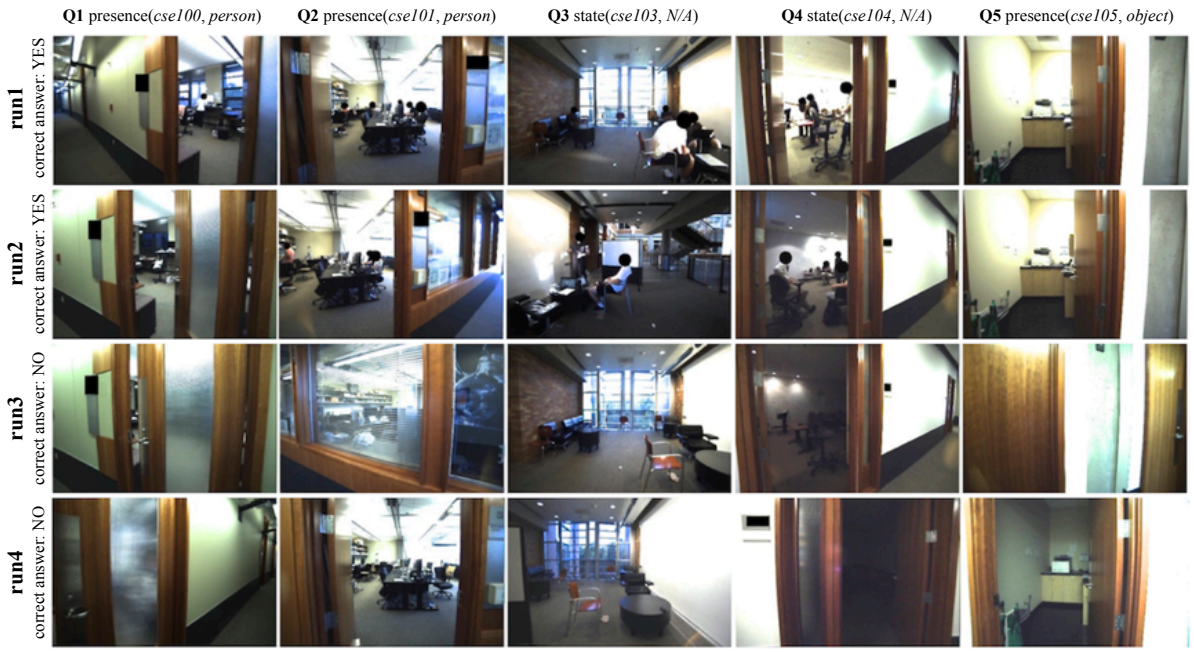


Figure 3.9: Returned images from the experiment I runs. The first two row show images from the runs with the ground truth answer "yes" and the next two rows show images from the runs with the ground truth answer "no" for the corresponding checking questions (columns). The column header displays the best fit information descriptors for the corresponding checking questions (Q1–Q5).

target type annotations were acquired by a manual annotation process. These annotations were transformed to the information presence term using the height-based conversion function with the height mean $\mu = 1.65$ and standard deviation $\sigma = 0.05$. For the visibility term, the horizontal and vertical field of view of the camera were discretized into a 100×100 grid for raytracing, and used $\theta = 15$.

Experiment I

We evaluated our viewpoint estimation algorithm's ability to deliver images that can be used for answering checking questions on questions frequently asked during the initial deployment

(Sec. 3.4): [3]

Q1. *Is {person} in his/her office?*

Q2. *Is there anyone in the mobile robotics lab?*

Q3. *Is the breakout area occupied?*

Q4. *Is the conference room occupied?*

Q5. *Is there a stapler in the printer room?*[4]

Note that all questions were yes/no questions. The corresponding best fit information descriptors for the questions are shown as the column headers in Fig. 3.9. We ran the viewpoint estimation with the iterative refinement four times throughout a day for each checking question. We choose the timing of the run so that the ground truth answer for two runs were "yes" and the other two runs were "no". However, we did not control the visibility and reachability conditions of the target locations to capture natural variations in the building environment. Fig. 3.9 shows the returned images from each run and Fig. 3.11 shows the details of the viewpoint estimation algorithm for Q2 and Q3 runs.

Viewpoint quality. To understand potential users' ability to extract answers from images chosen by our viewpoint estimation methods, we conducted a user study with 10 building occupants. For each image returned from the runs described above, we asked participants to respond to the corresponding checking question (Q1-Q5) based on the image. Response options were "definitely yes", "probably yes", "I don't know", "probably no", and "definitely no". Fig. 3.10 shows the results from the user study. We consider a response to be correct if a user responds with "definitely yes" or "probably yes" when the ground truth answer is "yes" or if they say "definitely no" or "probably no" when the ground truth answer is

[3]Except the "Is there a stapler in the printer room" question, which is a substitution for the "Is there free food in the kitchen" question.

[4]Equivalent to "Is there free food in the kitchen?"

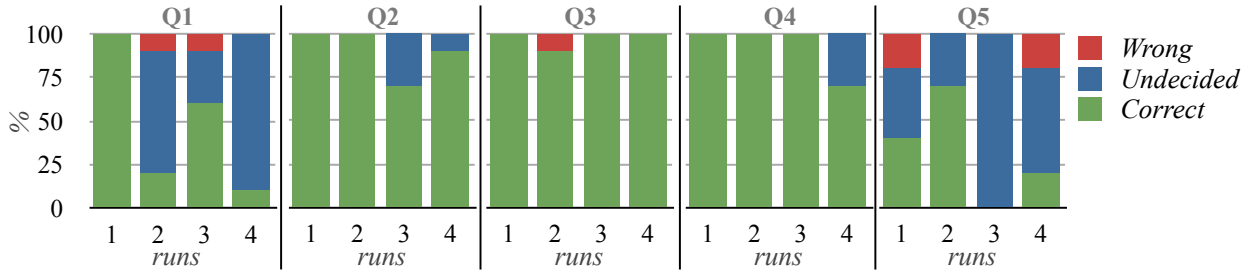


Figure 3.10: Distribution of answers generated by user study participants from images captured by the robot.

"no". We consider a response wrong if the user's answer contradicts the ground truth and undecided if the user responds with "I don't know."

Overall participants achieved a high classification accuracy, particularly for questions Q2, Q3, and Q4. It can be observed from Fig. 3.9 that high "I don't know" rates and non-zero wrong response rates are due to the limitations of the sensors or the encountered situation (e.g. target locations blocked by closed doors or bad lightening conditions) rather than a limitation of the algorithm. For example, in the 4th run for Q1, 90% of participants said they were undecided if there is a person inside the office because they observe the door being closed; not because the robot did not provide sufficient information. In other words, if the users were to try and answer Q1 in this situation by going to the target location themselves, they would reach the same answer through passive observation. Similarly in the third run of Q5, all of the participants indicated that they did not know if there was a stapler in the printer room because the door to the room was closed.

In the other runs of Q5, the wrong and undecided answers are due to the difficulty of seeing the stapler in the small and reduced-quality image (due to lighting). This problem could be mitigated by allowing the robot to navigate into the room to obtain a better viewpoint, post-processing images to enhance color contrast, or allowing participants to zoom in on parts of the image to obtain the answer.

Handling dynamic changes. Fig. 3.11 illustrates how the viewpoint estimation algorithm

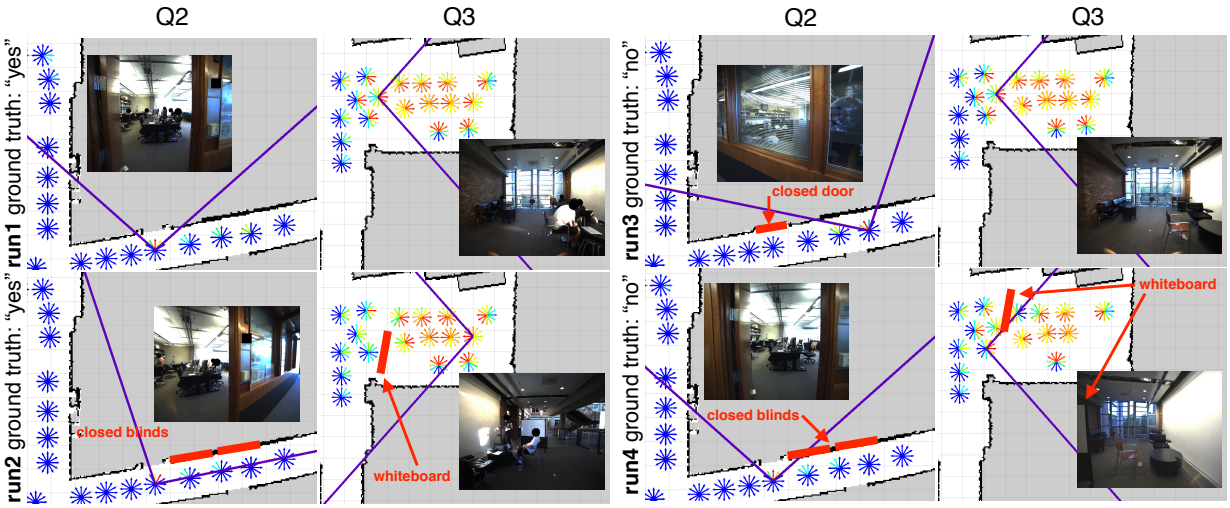


Figure 3.11: Viewpoint estimation details for Q2 and Q4 in Experiment I. The evaluated quality of each viewpoint, based on $P(\mathcal{I} = 1|v, s; M, D)$, are displayed as colored arrows over the 2D map. Warm colors (red) indicate greater quality. The camera field of views of the selected optimal viewpoint are drawn with the purple lines and the image captured from this viewpoint is shown. The dynamic changes to the environment that influenced the viewpoint estimation algorithm are annotated in red.

adapts to dynamic changes in the environment by selecting alternative viewpoints with similar information content. For Q2, the robot was able to capture the view of the lab's inside through its door when it was open, but also through its window when the door was closed and the blinds on the window were open (run 3). Similarly for Q3, the robot navigated to the other end of the breakout area and turned around to capture the view of the area, when it encountered a whiteboard blocking the view from its initial viewpoint.

Experiment II

Next, we considered the retrospective querying scenario in which the viewpoint estimation is performed on previously collected data. This captures scenarios that involve questions concerning the past ("Was Mike Chung in the robotics lab?") where the system attempts to provide a response based on incidental visits to a place while performing other tasks involving navigation (e.g. patrolling or delivery). We considered four such cases. In the

first two, the robot was navigating near the breakout area as shown in the left column of Fig. 3.12 and the viewpoint estimation was later used to answer the question Q6: "Was the breakout area occupied?". In the latter two cases, the robot was navigating near the conference room as shown in the right column of Fig. 3.12 and the viewpoint estimation was later used to answer the question Q7: "Was the conference room occupied?". In all cases, the viewpoint estimation algorithm was used with the latest 3D occupancy map available from the collected data, within the constrained search space of *visited* viewpoints.

The retrieved images and details of the runs for this experiment are shown in Fig. 3.12. We observe that the viewpoint estimation algorithm produces appropriate responses in the retrospective question answering setting. In response to Q6, the robot needs to capture the breakout area from a set of candidate viewpoints that are tangential to the area (i.e. the robot went by the breakout area without looking towards it). We see that the algorithm selects viewpoints that are *further away* in the path such that the target area can be captured on one side of the robot's field of view. In run 1, the robot is able to choose viewpoints that are further from the target area than in run2, and hence captures more of the area by exploiting the fact that the bridge-like corridor does not have walls obscuring the robot's view of the target area. In response to Q7, the robot is able to capture a larger part of the target conference room by choosing viewpoints near two different doors to the room in the two different runs where the robot was navigating in opposite directions.

3.9 Discussion

3.9.1 Non-checking Information Gathering Types

Although this chapter focused on information checking, our formative studies indicated that other types of information gathering might be useful. During the WoZ deployment experiment (Sec. 3.4), non-checking questions were rejected; however, users still asked those questions. For example, users made *search* requests, e.g., "Is there an empty conference room in the Computer Science building?" and "Which meeting room has the best visibility of

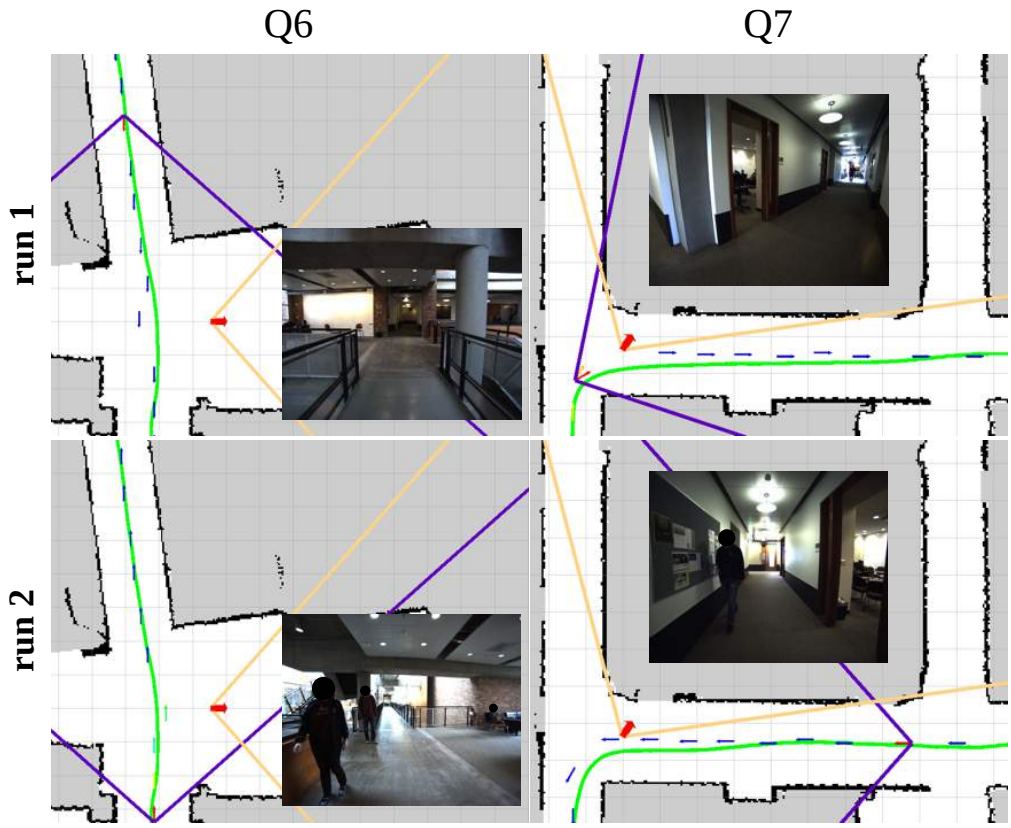


Figure 3.12: Experiment II results. The path taken by the robot is shown with the green line and the evaluated quality of viewpoints along this path are displayed as colored arrows. Warm colors (red) indicate greater quality. The camera field of view of the selected viewpoint on the path is shown with the purple lines and the image captured from this viewpoint is provided. For reference, the optimal viewpoint that would have been selected if the robot were to navigate back to the scene to capture the requested information is shown with the orange lines.

Mount Rainier today?" We also observed *monitoring* type questions, such as, "Has {person} arrived yet today in the CS building?" One respondent described the desired *summarization* capabilities:

> *I would love to be able to ask about current building statistics, such as what lights are on/off, which projectors are powered on/off, what the building internet up/down bandwidths currently are, current temperature(s), power usage, water*

> *usage, etc. Also, if the robot could do some kind of mood recognition whenever it saw a face, like "happy" or "sad," being able to ask about how the average mood in the building is today would be really cool.*

As in this example, many others wanted a system that combines the ability to check local state with other types of information that is already available through other sources, such as seminar schedules, weather, or nearby coffee shops.

3.9.2 Privacy Concerns

During the deployment experiment, we wanted the information gathering service provided by the mobile robot to be as similar as possible to humans gathering the information themselves. We displayed the name of the person for whom the robot was gathering information (Sec. 3.4.2) and announced the question before taking the picture. We plan to further explore strategies for mitigating privacy concerns by using the robot's embodiment to communicate the primary user's intent.

3.9.3 Views of Multiple Stakeholders

During the WoZ deployment and real-world experiments (Sec. 3.4, Sec. 3.8.2), several non-primary stakeholders expressed their opinions. The building operations manager expressed potential concerns regarding using an InfoBot over the long-term because of how the robot delayed elevator traffic. Indeed, we noticed the robot frequently held up the elevator for a long time because it failed to catch or leave it due to a navigation failure. Some bystanders expressed privacy concerns despite our efforts to alleviate them (Sec. 3.9.2). For example, we received emails requesting more transparency in the type of data the robot was collecting while navigating and regarding the robot's hours of operation. We received some opinions from building occupants from whom information was requested, as well. Some found the way the robot announced it was taking a picture to be acceptable but preferred to answer the question directly, e.g., verbally to the robot. We believe that better understanding

requirements from multiple stakeholders is critical and plan to investigate this topic further in the future.

3.9.4 Limitations

WoZ Deployment Experiment

Although our studies indicated that building occupants found InfoBots to be useful, the influence of the novelty effect in these results cannot be disregarded. For example, during the deployment experiment, the building occupants who were the subjects of the requested information seemed to be mostly amused. We have not explored any potential variations of the user interface for the primary users. We used a fixed user interface set up–our choice of using web interface consisted of free form text input and text and image response, as well as email notification–to conduct a consistent experiment. In the open-ended feedback on the post-deployment survey, some commented on interface elements and requested more feedback about the InfoBot's progress after submitting a question (current question queue or estimated time of response).

Information Checking Framework

We assume that the user's question mentions a single target location that can be feasibly captured from a single viewpoint. We consider a question such as "Is Mike Chung in this building?" as a *search* type question, and therefore out of scope for our framework. However, one can imagine a search method that embeds our approach for checking information at multiple target locations, within a larger planning framework. Similarly, questions that mention multiple target locations, such as "Is Mike Chung in the robotics lab or his office?" are not handled by our natural language component; however, this task could simply be considered as two separate information checking requests.

Another limitation is that human users need to extract the answer to their own question from the provided image, rather than receiving a definite answer. Although this part of the

task could also be automated with recent image understanding methods, we chose to leave it to the users since they can perform image understanding tasks robustly and efficiently [170]. Finally, our work focused on capturing information from a single image while images from multiple viewpoints or multiple images from the same viewpoint (to capture dynamic events) could potentially provide answers to a richer set of questions.

3.10 Conclusion

This chapter investigated an indoor information-gathering service for mobile robots. Specifically, we first categorized indoor information-gathering task types, conducted a formative study, and shared (1) our survey findings on people's expected usage of InfoBots, and (2) empirical findings of people's actual usage of InfoBots. We presented a framework and an implemented end-to-end system for answering natural language questions from users about the robot's environment. Our system evaluation, based on the questions collected from diverse inhabitants of the building in which the robot was deployed, shows the feasibility of realizing a mobile-robot based information-checking service.

Chapter 4

EXPLORING THE USE OF ROBOTS FOR GATHERING CUSTOMER FEEDBACK IN THE HOSPITALITY INDUSTRY

4.1 Background

Gathering customer feedback is a critical component of the hospitality industry. Hotels have long amassed guest feedback to measure customer satisfaction and loyalty as well as staff performance. The collected feedback helps hotels to monitor service quality, make necessary improvements, and ultimately stay ahead of their competition [227].

We believe service robots can be an effective medium to elicit and gather guest feedback in hotels. Robots draw people's attention in public spaces [33, 69, 101, 190, 144], and their interactive behaviors can be precisely controlled to enforce hotel brand standards or elicit certain emotional responses [179]. At some hotels, robots such as the Savioke Relay, are already interacting with guests, e.g., by delivering small items to guests, which gives them opportunities to solicit guest feedback.

Exploring a new real-world application for service robots is not trivial. Deploying robots in workplaces may require structural and procedural changes in workplace design and use [146], consequences that are difficult to foresee and costly to discover post-deployment. Also, it is difficult to gather quality feedback from potential users in up-front research sessions because most people lack firsthand experiences with robots in their workplace [155].

In this chapter, we address the following two research questions: (1) Can we use robots to gather feedback from hotel guests? (2) How should we design robotic systems to gather better customer feedback? The answer to the first question depends on the context in which robots interact with customers; hence, it is important to understand the physical and situational context as well as customer opinions. It is also important to understand the needs of service

industry workers and their current practices for gathering customer feedback. Finally, to answer these questions through field deployments, it is critical to capture the context in which the customer experiences the service and decides whether and how to respond to a robot's solicitation for feedback. To that end, our research focused on the Savioke Relay robot, which was deployed in approximately 70 hotels in January 2018. Using a mixed sequential exploratory research approach ([45]) adapted for the robotics domain, we present four studies exploring customer feedback collection for robots in the hospitality industry. Our studies involve multiple real-world stakeholders, such as hotel employees and potential guests, and long-term real-world deployment studies ranging from 3 weeks to 4 months. We discuss the overall approach in Sec. 4.2 and share our experiences and learned lessons as findings and design implications from each study in subsequent sections (Sec. 4.3, Sec. 4.4, Sec. 4.5, Sec. 4.6).

4.2 Overview of the Approach

We took a mixed sequential exploratory research approach [45] with the following steps:

1. *Need finding interviews* with hotel management (n=5) at five hotels that already used a Relay robot for guest room delivery. We explored the current practices hotels use to gather customer feedback and the contexts in which the robot could gather feedback (Sec. 4.3).

2. *An online survey* with varying-frequency hotel customers (n=60). We explored guests' perceptions of the Relay room delivery robot and their willingness to respond to the robot in different situations vs responding to other feedback methods (Sec. 4.4).

3. *Passive observations and follow-up interviews* (n=5) at three of the five hotels in (1). We deployed the prototypes of robot-based customer feedback applications for 3-4 months to explore the value that the robot added, interaction patterns with the robot, and real-world challenges in gathering feedback from customers with robots (Sec. 4.5).

Table 4.1: Robot Actions

Participant job title (ID)	**Used Relay since**	**Number of rooms**	**Location**
General Manager (P1)	7/1/2015	62	SF Bay Area
Director of Operations (P2)	3/2/2015	172	SF Bay Area
General Manager (P3)	4/1/2017	175	SF Bay Area
General Manager (P4)	8/25/2015	231	SF Bay Area
Guest Satisfaction Manager (P5)	7/1/2015	304	Los Angeles

4. *Passive observations, follow-up questionnaires, and analysis of measurements* in a three-week deployment at a kitchen area of the Savioke headquarters. We explored the role of different robot behaviors (mobility and social attributes) in gathering feedback and identified design constraints (Sec. 4.6).

4.3 Need Finding Interviews at Hotels

The goal of our need finding study was to learn (1) current practices for collecting guest feedback, (2) opinions on the idea of gathering feedback with the robot, and (iii) guest experiences with using the robot at their hotels.

4.3.1 Participants

We selected a total of five participants who have an administrative job for the five hotels that had been using a Relay robot. To capture the various perspectives, we chose participants with different positions in the hotel, with the hotels varying in size and location (Table 4.1). We intentionally chose participants with firsthand experience interacting with a robot at their workplace; these participants did not have technical jobs because we wanted to avoid interviewing people who have completely unrealistic expectations or are too pessimistic about the robot abilities [155].

4.3.2 Procedure

We interviewed the participants regarding existing methods for collecting customer feedback and current usages of the Relay robot at their hotel. The interviews were conducted at a place in the hotels where the participants felt comfortable, except P4 with whom we had a phone interview due to the location of the hotel. Each interview was structured as follows (see [39] for the actual interview protocol we used in our study):

1. **Introduction** of the interviewer, the purpose of the study, followed by consent for voice recording.

2. **Warm-up questions** about the participants, such as their roles at the hotel, their favorite part of their jobs, and how long they have been in the hospitality industry.

3. **Current practices for collecting customer feedback**, including whether participants have tried technology-based solutions, and how guest feedback is used. Probing questions include asking about pain points and asking for reasons behind comments (i.e. "Why?" questions).

4. **Participants' experiences with Relay** and their observations regarding customers interacting with the robot.

5. **Participants' opinions on collecting guest feedback with Relay.**

6. **Tour** of the hotel.

7. **Wrap-up**; a final question and answer session.

Throughout the interview, participants were encouraged to lead the conversation and were asked about memorable incidents if applicable. The interviews lasted between 30 minutes and 2 hours.

Note that our interview protocol includes a step for directly asking the interviewees about using the robots to collect feedback (5). This is not a conventionally done in need finding interviews. However, we included this question because we wanted to learn participants' perspectives and potential constraints for this use case.

4.3.3 Findings

The recordings from the interviews were transcribed by the author. We then conducted an inductive content analysis on the transcriptions and organized the participants' responses into the following themes: Existing Guest Feedback Collection Methods (Sec. 4.3.4), Service Recovery Strategies (Sec. 4.3.5), Factors influencing Robot Usage (Sec. 4.3.6), Participant Comments on Collecting Guest Feedback via Robots (Sec. 4.3.7).

4.3.4 Existing Guest Feedback Collection Methods

All participants reported that they used the brand-required, post-stay survey and accessed TripAdvisor ([174]) to learn what their guests thought about their hotel experience. Another commonly mentioned feedback collection method was having the hotel staff directly ask guests about their stay at likely points of interaction, such as when guests came to the front desk to ask a question or to checkout. P2, P4, and P5 mentioned that they train their staff members to elicit feedback from guests whenever an opportunity arises. P4 further reported that their staff members must ask whether a guest needs anything before closing a conversation as a part of their brand standard.

Two participants shared their experience with a more recent, mobile phone-based instant messaging solution for communicating with guests([16, 108]). P5 noted that the real-time aspect of the solution helped them to identify a few unhappy guest before they checked out. One downside P5 mentioned was the difficulty of informing guests about the availability of this service. P2 mentioned the messaging solution helped them better understand their guests' needs; however, it greatly increased front desk staff's workload, so eventually the hotel stopped using it. Both participants mentioned that today's guests prefer using a

mobile phone to having a face-to-face conversation, which was the main reason their hotel tried the mobile-based solution.

4.3.5 Service Recovery Strategies

Although we asked how the hotels gather guest feedback, the participants also explained their service recovery strategies, that is, strategies for returning dissatisfied customers to a state of service satisfaction. P2 reported that they respond to all the guests who participated in the post-stay survey. P3 and P5 reported that they respond to negative feedback directly on TripAdvisor to reduce the risk of losing future customers. P5 further explained their strategies to follow up on guest complaints based on guest type and problem severity. Other examples of recovery strategies include giving guests a 10% discount on their bills or offering a free dinner at the hotel's restaurant.

Some hotels emphasized the importance of getting guest feedback on-site. P2 and P5, who has experience with the mobile phone-based feedback collection methods (Sec. 4.3.4), elaborated on why collecting guest feedback on-site, esp. negative ones, will be helpful:

[P2] *Anytime we have the ability to capture the moment before they leave; that's when we can fix it. That's when we establish contact. [..] bring in the human, recover the guest, and make sure they leave as a happy customer.*

[P5] *[..] if after they've left, that's when they're telling me, guess what, I can't put two pillows now or I can't say "I'm gonna offer you a 10% discount." But if they can tell me while they're on property [..] I can apologize and send them two pillows and say "By the way, would you care to have dinner on us at the restaurant?" So it ties into knowing what's happening at the hotel in real time and be able to offer a solution.*

They also remarked on the irreversible impact of customer dissatisfaction:

[P5] *[..] after they left, they'll go on social media and let you know; which is not effective for the hotel. Because you didn't have the opportunity to fix it. Yes, you can fix it for the future but you can't fix it for that guest. [..] Everybody out there, all your potential customers are*

seeing this feedback.

P3, on the other hand, focused on the importance of positive ratings. P3 believed that the robot increased the chance of garnering positive feedback from customers.

[P3] *It's really a novelty item, more than anything else. [..] People like it and it is functional.*

4.3.6 Factors Influencing Robot Usage

Weekday/Weekend, Seasonal Influences

While describing their experiences with the Relay robot, participants shared observations about robot usage patterns in their hotels. P1 and P2 emphasized the usage difference between weekdays and weekends. Both hotels predominantly have business travelers on weekdays and local leisure travelers on weekends. Both reported greater use of the robot on weekends by leisure travelers. They explained that weekend guests spend more time in the hotel and show more interest in the robot, e.g. by taking a photograph with it and requesting a room robot delivery for fun. P3 and P4 reported seasonal differences in usage. During summers they observed an increased number of robot delivery requests due to increased occupancy rates (greater than 90%).

Guest Type

Although a diverse clientele routinely interacts with the Relay robot, all participants remarked on children users. P5 reported that they frequently see children hugging or following the robot. P4 shared a story of parents who visited the hotel expressly to surprise their children with the robot room delivery. In addition to children, P1 mentioned that some older adults take interest in the robot (e.g. taking a photograph with it) while others seemed leery of it. P1 also mentioned that people who work at technology companies are more interested in the robot and ask questions regarding its functionality and price. Finally, when we asked

if they noticed a type of guests who do not pay much attention to the robot, P1 and P2 mentioned business travelers and people who are traveling alone.

4.3.7 Participant Comments on Collecting Guest Feedback with the Robots

Participants had mixed feedback about the idea of using the Relay robot to collect guest feedback. P2 and P5 responded positively to the idea. P2 mentioned that were the robot able to report feedback in real-time, it would be valuable for the hotel staff, who would then would be able to resolve issue before the customers check out. P5 predicted that using the robot would increase the chance of eliciting guest feedback since many guests do not report problems to avoid hurting a staff person's feelings. P5 suggested that the robot could be an effective, neutral, middle person.

[P5] *We would get more feedback. Because a lot of customers do not like face to face interaction. They feel, "Now I'm putting somebody down" or "I'm going to get someone in trouble." That's why we get so many hits on social media.*

P1 and P3 were skeptical about the value of guest feedback data gathered by the Relay robot. They pointed out that the robot is not capable of collecting rich feedback due to its small screen size. However, they still wanted to try because guests want more interactive robots, and feedback gathering would increase robot utilization. P4 mentioned that gathering guest feedback via the robot would not be valuable since they already have other means to collect this feedback; further, P4 noted that the hotel already extensively uses the robot for delivery tasks, so they are hesitant to add to its workload. In addition, P4 was concerned that the guests receiving the items from the robot are often children.

Participants gave some suggestions about when and how the robot should attempt to gather customer feedback. P1 suggested providing the option of completing a general satisfaction survey, through a link that would say "would you like to rate your overall stay?" after each delivery. P2 wanted the robot to conduct a short survey after delivering receipts to people in the restaurant. They also suggested having the robot hand out discount checks

if they fill in the survey. P5 wanted the robot to move around in the lobby area and solicit feedback from passing guests. P5 suggested having guests report problems to the robot and rate the severity of the problem. P5 also suggested having the robot display compensation options to unhappy guests and attempt to resolve problems by itself. P3 suggested integrating TripAdvisor and the robot so guests' positive reactions could be reflected on the hotel's profile on TripAdvisor in real-time, while the guests interacting with the robot.

4.3.8 Design Implications

Robots could play a role in helping hotels disseminate positive customer feedback widely. They could also help to identify dissatisfied customers while they are still on-site. To this end, robots should take advantage of being in the context of the service to encourage customers to express their opinions in the moment and on the property. They should be designed to make guests feel more comfortable giving negative feedback, acting as neutral liaisons between guests and the hotels. Robots should respond to customer feedback, possibly by attempting to recover from service failures without human intervention.

By default, interactions for gathering feedback should be short and, if possible, entertaining to accommodate the short attention spans of modern customers. Robots could leverage their status as *novelty items* to engage customers and encourage them to respond to questions. Ultimately, robots should adjust their strategies for eliciting engagements and interacting with customers based on the types of customers. For instance, robots could identify the type of a customer from initial interactions to decide which questions to ask and how to most meaningfully interact with the customer [116].

4.4 Online Guest Scenario Survey

Viewpoints of guests are important to hotels as well as to the robotics company providing the service. Hence, we conducted an online survey that explored potential guests' attitudes and motivations towards robot-based feedback solicitations compared to other solicitations.

Figure 4.1: (Left, Middle) Pictures of the kiosk and the Relay robot used in the online survey to illustrate Kiosk and Robot FD. (Right) A screenshot from the video that demonstrates room delivery to convey Robot RM.

4.4.1 Survey Design

To help participants contextualize the decision about responding to different kinds of customer feedback solicitations, we provided a motivating scenario. We chose one negative and one positive guest experience scenario adapted from the scenarios[1] commonly used in hospitality research [120]. Each participant read one of the two scenarios. We instructed participants to assume the situation described in the scenario had just happened to them. We asked them how likely they would be to respond to each of the following feedback solicitations (1: Extremely unlikely to respond; 5: Extremely likely to respond):

- **Email:** You received an online survey after you left the hotel.
- **Kiosk:** You noticed a kiosk near the front desk which says "How is your stay?"
- **Robot Front Desk (FD):** You noticed a robot near the front desk which says "How is your stay?"
- **Robot Room (RM):** You ordered a snack from the front desk and a robot delivered

[1]For details, see the Supplementary Materials of [40])

the snack to your room. After handing off the snack, the robot asks "How is your stay?"

We included an optional open-ended question asking for an explanation of each response. For Kiosk, Robot FD, and Robot RM, we included explanatory images and a video to give participants an accurate sense of what the kiosk and robot would look like (Fig. 4.1). Note that the four solicitation methods were selected based on Savioke's and their customer hotels' business interests.

4.4.2 Participants

Participants were recruited through Amazon Mechanical Turk. After agreeing to participate, online participants were directed to the single-page form that contained the scenario and the four questions about the four feedback alternatives. To control for quality of responses, we did not allow a person to participate in our survey more than once and rejected people who incorrectly answered the question used to identify those who randomly selected answers. We offered \$0.01 for participation and continued recruiting until we had 30 responses for each scenario. A total of 60 people (22 M, 38 F) responded in less than 2 weeks. Their age groups distributions were: 10% in 19-24, 30% in 25-34, 30% in 35-44, 30% in 45-54, 16.67% in 45-54, 10% in 55-64, 3.33% in 65-74. Respondents' answers to the question regarding the frequency of staying at a hotel in any given year ranged from 1 to 20, with a median of 2 times a year.

4.4.3 Findings

The means and standard deviations of the responses across the four solicitations were: M = 3.58 & SD = 1.43 (Email); M = 3.38 & SD = 1.58 (Robot RM); M = 3.25 & SD = 1.67 (Kiosk); and, M = 2.68 & SD = 1.69 (Robot FD). The same statistics across the two scenarios were: M = 3.22 & SD = 1.52 (Positive); and, M = 3.23 & SD = 1.72 (Negative). Fig. 4.2 shows the distribution of responses for each feedback solicitation method in each scenario. We conducted open coding for the open-ended question responses. We summarize

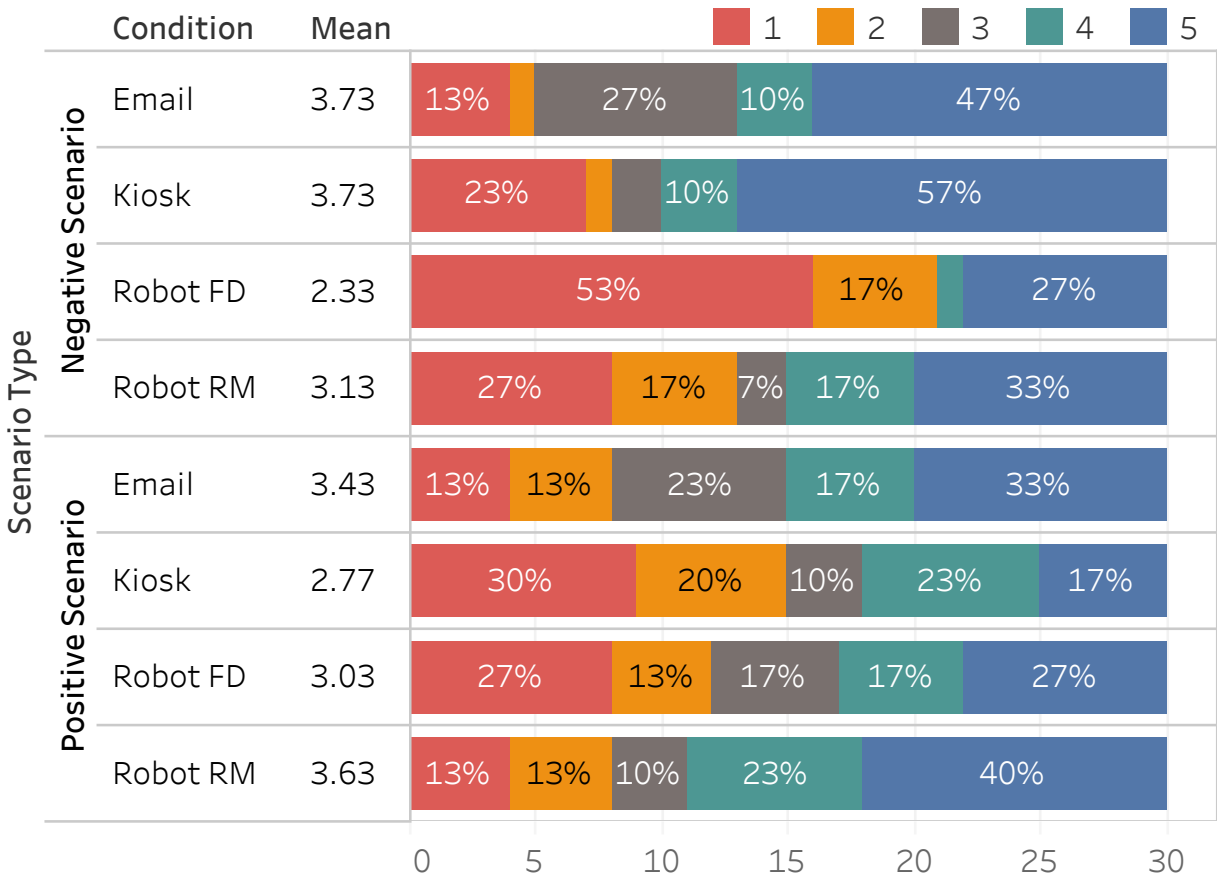


Figure 4.2: The distribution of responses for different feedback solicitations in positive and negative guest experience scenarios (1: Extremely unlikely to respond; 5: Extremely likely to respond).

our findings below.

Factors Influencing Guests' Willingness to Respond

The most frequently mentioned reason for responding to a solicitation method was for participants to share hotel experiences (e.g., *"I would want the hotel to know that my room was unacceptable."*). The second most mentioned reason was convenience. This was also the main reason why Email and Robot RM were rated highly. For example, a participant mentioned, *"If I'm already interacting with the robot, I may as well answer the question."* In fact, inconvenience was the top reason for participants' unwillingness to respond to a solicitation method.

Some participants had privacy concerns about using the Kiosk and Robot FD, e.g., *"Less likely, but I'd feel the staff might be watching over my shoulder."* They mentioned that they liked the Email and Robot RM methods because of increased privacy, e.g., *"I would respond because it is still kind of private and (the robot is) not out in the busy lobby."* Given our findings from the need finding interviews (Sec. 4.3), we noticed an understandable conflict between hotels and guests: hotels want more data from guests, but guests value privacy want it to be respected.

Influence of the Scenario Type

The influence of the scenario type varied across the four solicitations. When the participants read the *positive* scenario and were presented with Email or Kiosk, they were not enthusiastic about informing the hotels (Email: $p < 0.001$, Kiosk: $p < 0.00001$; one-tailed, paired t-test); more than half said they would be unlikely to respond (i.e., responded with < 3) because they would not have the time or did not want to further engage with the hotel. In contrast, more than half of the participants who read the *negative* scenario and were presented with Email, Kiosk, or Robot RM would inform the hotel about their stay (responded with > 3).

The participants were more enthusiastic about responding to the two methods involving robots when they read the positive scenario than when they read the negative one (both

Robot FD and Robot: RM $p < 0.0001$; one-tailed, paired t-test). Some participants would be likely to respond to the robot (i.e., responded with > 3) predicted that interacting with it would be pleasant and would make their experience at the hotel unique. In contrast, the participants who read the negative scenario most commonly questioned the possibility of the presented scenario, for example, by commenting *If they had enough money for a robot, they would have enough money for a cleaning staff to do a good job.*

Perception of the Robot-based Solicitations

Regardless of the scenario type, participants mentioned similar reasons for liking or disliking the two robot-based solicitations. When participants were willing to respond, the reasons they offered included the novelty effect (e.g., "*I will fill out of curiosity.*"), the entertainment value of the robot (e.g., "*The robot would interest me and be a fun addition to my stay.*"), and the feeling of obligation (e.g., *"Because the robot needs a response."*). When they were not willing to respond, they mentioned that they disliked the robot (e.g., *"A feedback robot? That's just weird; the world isn't ready for that."*), did not trust the robot (e.g., *"I wouldn't trust the robot."*), or envisioned a potential difficulty using the robot (e.g., *"No clue how to interact with it."*).

4.4.4 Design Implications

The process of providing feedback to the robot should be as convenient as possible for the guest, for example, by keeping survey questions short or asking a question at the end of a different interaction to allow the guest to ignore the question. Robots should respect the privacy of guests even at the cost of losing data they can provide to the hotels since the goal of both hotels and robotics companies is to satisfy customers. For example, robots should collect guest feedback in private settings and restrict using sounds or movements that could reveal guest responses in open spaces.

Figure 4.3: (Left) The Relay robot running the breakfast survey behavior at the P1's hotel. (Right) The robot asking for employee feedback about the provided meal at the Savioke headquarters. The robot briefly stopped navigating to respond to the employee.

4.5 Hotel Deployments

Three interviewees (P1, P2, and P5) from the need finding interviews (Sec. 4.3) were interested in using a robot-based feedback solicitation at their (different) hotels. This gave us the opportunity to test our idea in the field and thereby better understand real-world challenges.

4.5.1 Breakfast Room Survey

P1 requested that we enable the Relay robot to ask a few customer satisfaction survey questions to the hotel guests in the breakfast room area. We collaborated with a robot UX designer at Savioke to prototype a breakfast room survey robot's behavior and refined it with P1 to meet hotel's requirements. The prototype behaved as follows. Upon launch, the robot navigated to a predefined location near the entrance of the breakfast room. The robot

then stayed in place and displayed the question, "Good morning! How was your breakfast?" together with a five-star rating response field. When a customer response was detected, it asked the second question, "How is your stay?" and responded on receiving five stars with a happy facial expression and a dance. If no response was received within five minutes, the robot played whistle sounds and displayed the two messages, "Hello! I'm Relay, a delivery robot." and "Need anything? Dial 0 from your room and I'll bring it to you!" in sequence to elicit attention from passing guests.

In May 2017, we installed the break room survey behavior to the Relay robot in P1's hotel as shown in Fig. 4.3 (left) and instructed all staff members there about how to start and stop the survey behavior and handle the potential problems. While the robot's status information, such as its location and remaining battery charge, was available on the web interface, the hotel staff was not able to see the customer response history (due to logistical reasons). However, P1 insisted on deploying the breakfast room survey behavior as is to increase utilization of the robot. To gain insight into how the hotel used the provided survey behavior, we monitored their usage for four months both remotely and through two on-site visits, on Tuesday and Saturday of the 3rd week of September 2017, to observe the robot in context. We were not allowed to talk to guests for logistical reasons.

4.5.2 Low-Ratings Alert for Guest Room Delivery

P2 and P5 requested that we enable the Relay robot to ask customer satisfaction questions after each guest room delivery and alert staff members on receiving negative responses from guests. We collaborated with a Savioke robot UX designer and prototyped a feature that adds the "How is your stay?" star rating question after a delivery confirmation interaction and sends email alerts on receiving ratings below three stars.

We provided the low-rating alert feature to both hotels in June 2017. We configured the feature to send emails to the staff mailing list and logged the usage of the feature for four months. On the first week of September 2017, we interviewed P2 and P5 and one staff person at each hotel.

4.5.3 Findings

Despite not providing actual feedback from customers, the staff members at P1's hotel used the breakfast room survey extensively. Over the four months, they used the survey every day except for three days post-deployment (123 days). On average, the survey ran for 229 minutes, and 43 questions were answered per day. On our two visit days, we observed approximately 9 guest-robot interactions on the first day and 22 on the second day. The robots received responses to the customer satisfaction question 1707 times at the P2's hotel and 709 times at the P5's hotel during the four months deployment. Of those, 46 and 17 (2.70% and 2.40%), respectively, received less than three stars.

Values of Robot-Based Guest Feedback Collection

As predicted in the need finding interviews (Sec. 4.3), P1 reported that the robot's ability to provide unique experiences to the guests was its most valuable aspect. P1 nonetheless acknowledged the potential benefit of the data collected by the robot: *"Getting a report that shows me the overall scores, that would be great. That way, at least I could track what days people are not happy with."* Regarding the low ratings alert feature, both P5 and P2 were satisfied with the feature and mentioned they were able to capture 2-3 unhappy customers per month.

Hotels Used Their Domain Knowledge

We learned that the time and location of running the breakfast survey behavior were carefully selected by the hotel staff. The robot was located in front of the breakfast room area, which was located right next to the elevator. Hence, the robot was seen by people going in and out of the breakfast room as well as guests waiting for the elevator; most people noticed the robot immediately or via the whistle sounds it played. We also noticed the network effect: whenever a guest started interacting with the robot, it raised the attention of the other passing or waiting guests. When we continued our observation at P1's hotel in the

afternoons, we observed almost no activities near the breakfast room.

At P2's hotel, a staff person mentioned they paid extra attention over the weekend because the hotel usually receives a higher number of complaints while the staff person from P5 reported that they paid extra attention to input from guests with a membership.

Privacy Issues

Due to logistical reasons, our two prototypes did not rigorously follow one of our own guidelines: the robot should respect customer privacy. For example, the robot responded with a whistle sound and a dance on receiving five stars while surveying in the breakfast area, which allowed the people around the robot to notice what rating the person had given. In addition, the low ratings alert feature did not give customers the choice of notifying the front desk when they responded with less than three stars on the "How is your stay?" question. We learned that the hotels took advantage or were unaware of the consequences. For example, P1 mentioned that they like to monitor the guest interacting with the robot to not only to identify unhappy customers but also to understand the status of the hotel in general by eavesdropping on nearby conversations around the robot. Regarding the low ratings alert, all four interviewees mentioned that they always followed up with guests to recover potentially unhappy guests. No one considered the guests who do want to be contacted by the hotels.

4.5.4 Design Implications

The user interface for the robot should be designed to protect guest privacy. For example, on receiving complaints, the robot should ask whether the guest is comfortable with its informing hotel staff about the complaints. To maintain the robot's position as a neutral liaison between hotels and guests; the interface should not reveal the guest's feedback if they do not want to inform the hotel. The interface for the robot should support using the hotel users' domain knowledge, e.g., providing an option to customize the messages used during survey, schedule survey behavior, or change survey location.

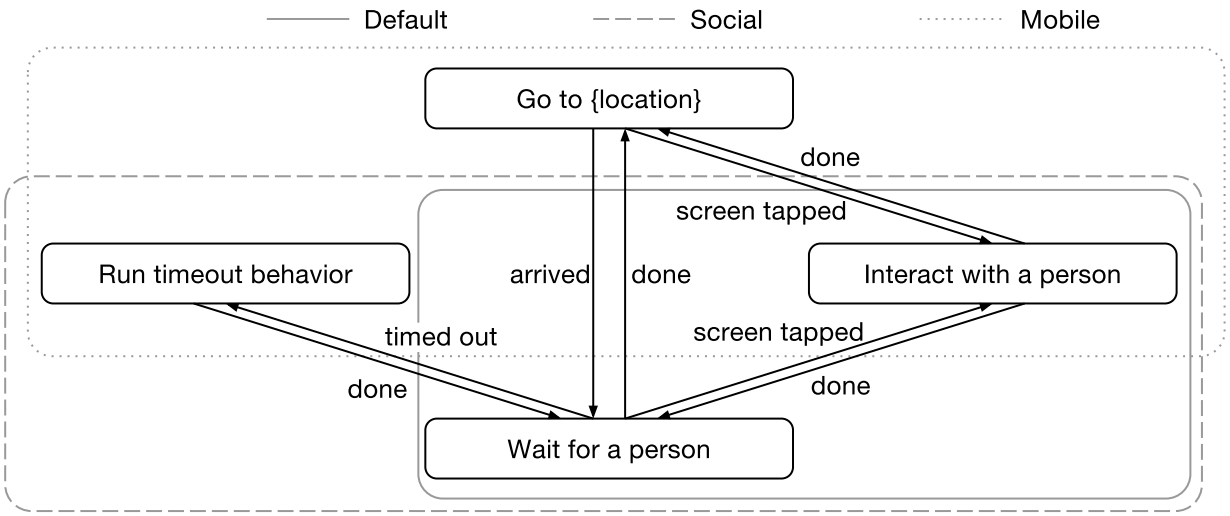


Figure 4.4: Finite state machines that implement the three robot behaviors: Baseline, Social, and Mobile.

4.6 Kitchen Deployment

As a final step in our exploration, we wanted to better understand: (i) the impact of the Relay robot's behavior on eliciting customer feedback, and (ii) the opinions of customers who are experiencing a service. In the hotel deployment study (Sec. 4.5), testing out different robot behaviors at hotels or talking to guests was not an option for logistical reasons. Therefore, we deployed a survey robot at the kitchen of Savioke's headquarters.

4.6.1 Study Design

We believe the Relay robot has two key properties that offer competitive advantages when gathering feedback relative to other feedback modes: *mobility* to move towards potential respondents, and *social agency* to increase the engagement using social cues [144]. By varying these properties, we designed the following three behaviors:

- *Baseline* was designed to mirror the experience of using a kiosk, like the Happy-or-Not Smiley Terminal [78]. The robot stayed at a predefined location and asked a question

about the meal using a five-star rating selection menu. We removed Relay's face, i.e., the eyes and speech bubble shown in Fig. 4.1, with a white background and did not use any sounds or movements. If a person answered the question with a rating, the robot responded by displaying "Thank you" for 5 seconds.

- *Social* was designed to make people perceive the Relay robot as a social agent. While its general behavior was similar to that of Baseline, the Social robot used more animated messages, including sounds, in-place movements, and LED light patterns. We kept Relay's face; all messages were displayed in the robot's speech bubble by using facial expressions. When nobody interacted with the robot for more than five minutes, the robot encouraged people to rate their meal with sounds and texts (i.e., 'Run timeout behavior' in Fig. 4.4).

- *Mobile* was designed to increase the chance of people noticing the robot by setting the robot in constant motion. The robot moved between two or three locations, encouraging people to leave their feedback. Once a person tapped the screen, the robot stopped and interacted with the person as in Baseline.

All experiments took place in Savioke's kitchen area, where the company provides meals to its employees at least three times a week. The kitchen area, approximately 700 square feet, was consisted of three sub-areas: a countertop area, a fridge area, and a table area (Fig. 4.5). Savioke had approximately 45 employees at its headquarters, who were mostly engineering and sales personnel.

To capture the situational variability, we conducted experiments during three company-provided meals: *Monday breakfast*, *Monday lunch*, and *Wednesday breakfast*. The two breakfasts were served on the countertop in the kitchen and delivered by an office administrator between 8:30 am and 9:30 am. The Monday lunch was served on the largest bar table in the kitchen and delivered by a catering company between 11:30 am and 12:00 pm.

To understand their employee-perspectives after they rated a meal, we asked them to

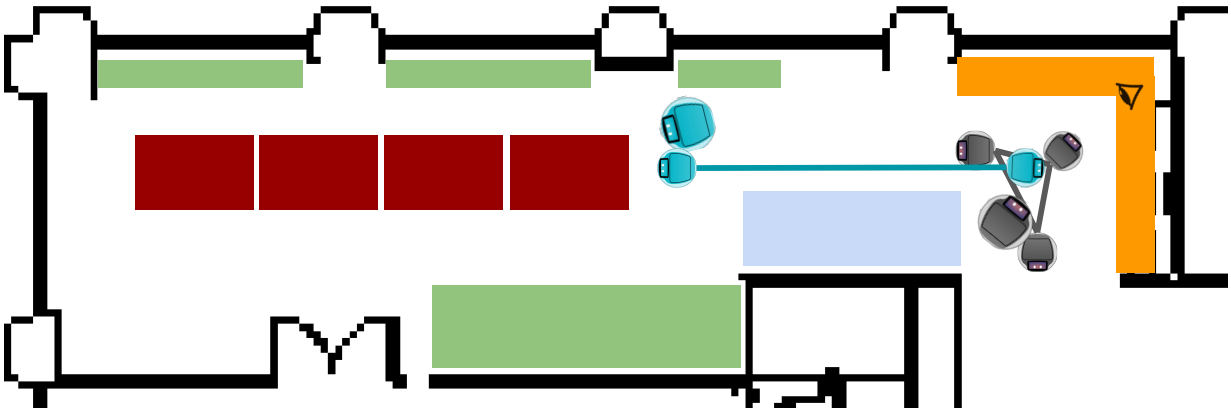


Figure 4.5: The layout of the kitchen where the Relay collected feedback about meals. Large icons show the poses of the robot in Baseline and Social behaviors (gray: breakfasts, blue: lunch). Small connected icons show the waypoints in Mobile behavior.

express their agreement (1: Strongly agree; 5: Strongly disagree) on a questionnaire with the following statements:

- **Effort:** Relay has made it easy to report my opinion.
- **Ease of use:** Relay was easy to use.
- **Perception of the robot:** Relay was pleasant to interact with and be around.
- **Change in perception of the service:** I felt better about my meal experience after reporting my opinion.

We included an optional open-ended question asking employees to explain their answers. The questions were adapted from the questionnaires used in related work [50, 81].

4.6.2 Procedure

The experiments were administered for three consecutive weeks, beginning the second week of August 2017. The order of conditions was counterbalanced using a Latin square design,

crossing the three robot behaviors with the three meal types. We worked with the office administrator to decide on the menu, selected to introduce variance as possible across the sessions.

Prior to the first experiment, we installed a Nest Cam Indoor camera and informed everyone in the company about the purpose and the duration of the study. We also shared our video recording policy and encouraged employees to share their privacy concerns. Each experiment session began when the food was served on the countertop or the bar table and ended when the food was taken away from the kitchen by the office administrator. The researcher started and stopped the robot behavior and video recording whenever a session was begun and ended. After each session, the researcher analyzed the recorded video and collected rating data to identify people who gave a star rating to the robot and send them the post-session questionnaire. We also encouraged all employees, whether or not they interacted with the robot, to directly email their feedback directly to the researcher.

4.6.3 Findings

During the three-week deployment, a total of 57 employees visited the kitchen area, and a total of 38 interacted with the robot. We determined the survey response rate by analyzing the recorded video and the data collected by the robot. We calculated response rate as $|M \cap R| \,/\, |M|$, where M is a set of employees who had a meal provided by the company and R is a set of employees who used the robot to report their feedback. We also reviewed answers to the open-ended questions, email feedback sent to the researcher, and video recordings of the sessions.

Response Rate

We compared the response rate across the three robot behaviors, the meal types, and the week numbers (Fig. 4.6a–c). We saw no significant impact of robot behaviors on the response rate; however, we saw influences of the meal types and the week numbers. The response rate dropped over the three weeks, potentially due to the novelty effect (Fig. 4.6b). The influence

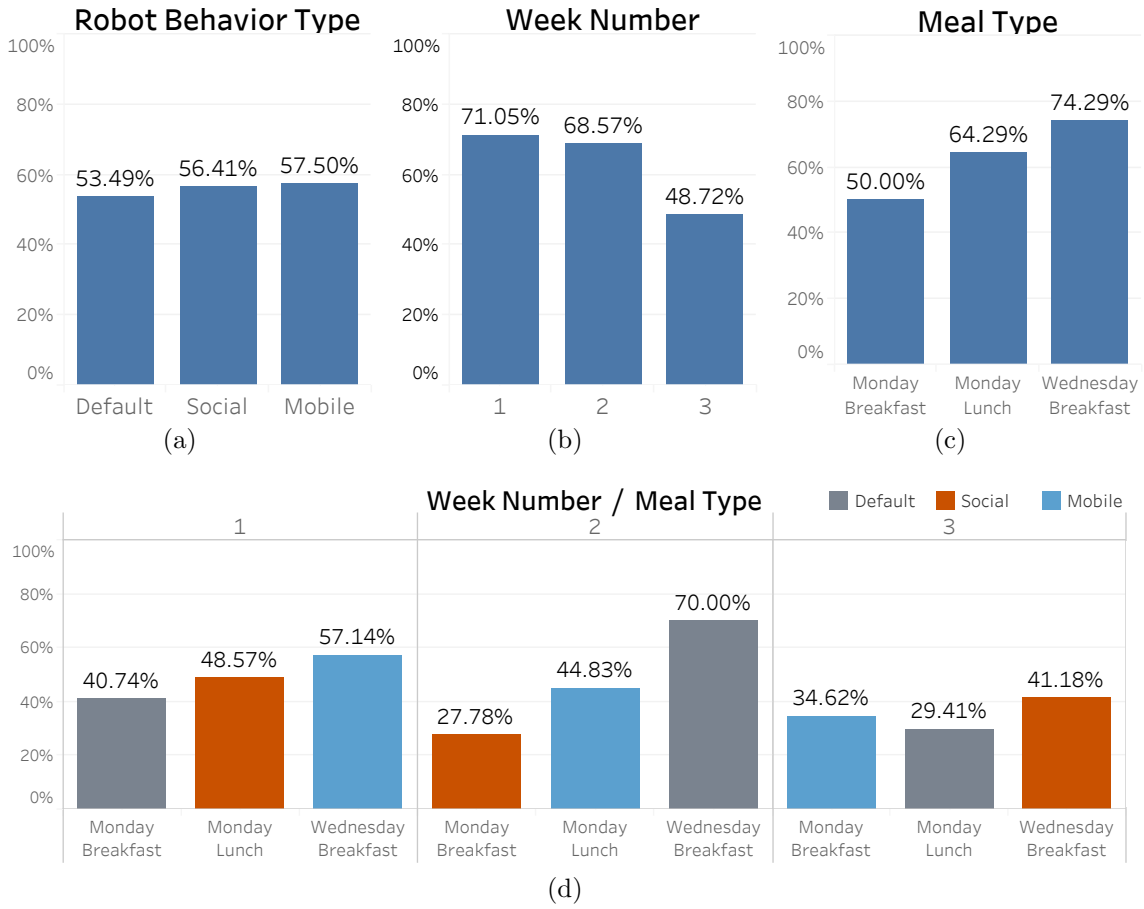


Figure 4.6: Response rate results in our three week deployment at Savioke headquarters of three different robot behaviors, accumulated by: (a) robot behaviors, (b) weeks, and (c) meal types; and response rates (d) broken down by session.

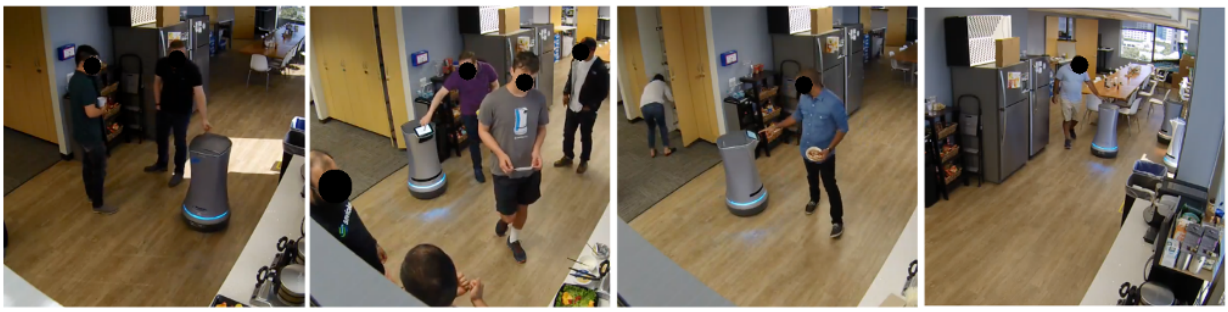


Figure 4.7: People interacting with the Relay robot running the Mobile behavior.

of meal type on response rate was the strongest (Fig. 4.6c), especially on the first and second weeks (Fig. 4.6d); we suspect that Savioke's organizational structure, e.g. holding of company-wide meetings on Monday morning, was the root cause.

Comparison of Robot Behaviors

Although the differences in response rates across the robot behaviors were not significant, we observed differences in how people noticed the robot, which can be considered the first step to interacting with it. People were more likely to notice the robot when it moved; 92% of employees in the kitchen noticed the robot running the Mobile behavior, but less than 58% noticed the robot running the Baseline and Social behaviors. The increased movement potentially contributed to the highest response rate (Fig. 4.6a) but decreased the usability for some employees; we observed five people who failed to tap the robot to stop its motion on their first try. We saw no significant differences between the Baseline and Social behaviors among our quantitative measurements.

User Perspectives

We received a total of 87 responses to the post-session questionnaire from 37 unique participants. The means and standard deviations for the post-session questionnaire were: M = 4.36 & SD = 0.94 (Effort); M = 4.67 & SD = 0.66 (Ease of use); M = 4.26 & SD = 0.96

(Perception of the robot); M = 3.46 & SD = 1.17 (Change in perception of the service). In answers to the open-ended questions regarding the effort, the most common positive remarks concerned the robot's ability to gather feedback in the moment, e.g., *"He was right there when the experience was still fresh in my mind."* The most common negative remarks were about the robot's inability to collect richer feedback, e.g., *"I wouldn't have been able to give concrete feedback other than to say it's good or bad."*

Regarding the ease of use, only three respondents who experienced the non-responsive touchscreen commented that the robot was difficult to use. Although we saw no negative comments regarding the perception of the robot from the open-ended questionnaire, one employee's email described discomfort concerning the mobile behavior: *"The robot in the kitchen is getting agitated (beeping more and more frequently as the morning goes on) and pushy (He started following me out of the kitchen and whether it was intentional or not, seemed to be trying to get me to take the survey.)".*

In the answers to the "Change in perception" open-ended question, some people did not find that the robot affected their perception of the provided service, e.g., *"The food is what matters".* They noted the lack of explanation on how their feedback would be handled, e.g., *"No confirmation on where the feedback is going, how it would be actionable."* They also appreciated the opportunity to give feedback, e.g., *"It seems good to know that I can provide input, whether it's good or not, to make things better next time."*

4.6.4 Design Implications

While using navigation movements could be useful, robots should avoid moving excessively so as not to disturb customers in the vicinity. Robots should have an option for guests to provide richer feedback. They should also clearly explain how the feedback will be handled or ask if/how the customers want followed-up contact.

4.7 Limitations

We acknowledge the following limitations in our research. First, we interviewed only five hotel employees in the need finding study, which is a small number of data points. In the online survey study, we did not consider a wide range of solicitation methods, such as those based on mobile phones mentioned by the hotel staff [39]. Finally, the kitchen deployment study was conducted at Savioke headquarters, the company that developed the Relay robot. Our findings may be biased due to the participants' familiarity with the robot and our design implications may not apply directly to the customer feedback collection use case in hotels due to the differences between office and hotel environments.

4.8 Conclusion

We presented four studies that explored the use case of collecting guest feedback with service robots in the hospitality industry. To account for the viewpoints of both hotels and guests, we administered need finding interviews at five hotels and an online survey concerning hotel guest experiences with 60 participants. We then conducted the two deployment studies based on deploying software prototypes for Savioke Relay robots we designed to collect customer feedback: (i) a hotel deployment study (three hotels over three months) to explore the feasibility of robot use for gathering customer feedback as well as issues such deployment might pose and (ii) a hotel kitchen deployment study (at Savioke headquarters over three weeks) to explore the role of different robot behaviors (mobility and social attributes) in gathering feedback and understand the customers' thought process in the context that they experience a service. Based on our findings, we summarize our design recommendations as follows. Robots should collect customer feedback in the context of the service and keep their interactions with guests brief to make feedback collection convenient for guests and useful for hotels. To protect the privacy of guests, they should be able to opt-out of sharing their feedback with hotel staff. They should also be able to provide more feedback if they so desire. The user interface for robots should support configuring the robot behaviors for the

robot users to use their knowledge about their organization to manage engagement levels of the robot.

Chapter 5

ITERATIVE DESIGN OF A SYSTEM FOR PROGRAMMING SOCIALLY INTERACTIVE SERVICE ROBOTS

5.1 Background

Service robots, such as the Savioke Relay, are becoming available in human environments such as hotels. It is important that, these robots not only be functional but also have appropriate, socially interactive behaviors. In this chapter, we investigate the research question, "How can we design a programming system for service robots to support socially interactive behaviors?" To do so, we took an iterative design approach. We first present results from a formative study with service industry customers. A key demand we discovered is that the robot must be aware of people present around it. We incorporated these lessons into the design of *iCustomPrograms*, a system for programming socially interactive behaviors for service robots, by extending an existing programming system *CustomPrograms*. Next, we performed two field studies with *iCustomPrograms* and iterated its design. In the first field study, which took place at an airport, we witnessed people initiating interaction with the robot in unanticipated ways. The second field study, which took place over 2 weeks at 5 service industry properties, evaluated the socially interactive applications created with *iCustomPrograms*. Our experiences and findings from each study show the usefulness of our system in the field. Importantly, they also provide insights for the design of future interactive applications for service robots and programming systems for creating such applications.

5.2 Formative Study

We used the Savioke Relay robot as the target robot platform in this chapter. Although the Savioke Relay robot was built for a specific application (room service in hotels), it can be

Table 5.1: Summary information about service industry properties studied in this chapter.

Property	**Type**	**Used since**	**Point of contact**	**Requested applications**	**Target areas**
A	Airport	2/2016*	Corporate executives, Customer satisfaction manager	People delight, Service recovery	Indoor garden, Baggage claim, Immigration hall
B	Hotel	1/2015	Hotel manager, Business consultant, Front desk supervisor	People delight, Mobile kiosk, Demo	Lobby, Bar
C	Hotel	6/2015	Guest service manager, Sales & marketing director	People delight, Service recovery, Mobile kiosk	Lobby
D	Hotel	7/2015	Hotel manager, Guest experience manager	Service recovery, Demo	Lobby, Breakfast area
E	Hotel	8/2015	IT manager, Area general manager	Mobile kiosk	Lobby

* Used since field study I

considered as a generic platform with a wider range of applications. The goal of our formative study is to discover potential applications that are desirable for existing Savioke customers and inform the design of a rapid programming system for creating those applications. To that end, we gathered information from five customers from the service industry.

5.2.1 Data Collection

Information about the properties we studied in this chapter is summarized in Table 5.1. Property A was an airport in Southeast Asia and the rest were hotels in the San Francisco Bay Area. Unlike the other properties, Property A had not used the robot prior to the first field study. Additionally, the target areas considered by Property A were larger ($\approx 4000m^2$) and more crowded than the areas considered by the other properties ($< 200m^2$).

We analyzed meeting notes and email exchanges between Property A and Savioke employees. The meeting notes were collected from two meetings held at Savioke headquarters

and one meeting held at the airport in 2015. During those meetings, Savioke employees surveyed the target areas in the airport and performed demos of the Relay doing deliveries. The two groups also brainstormed potential applications together.

For Properties B–E, we analyzed the field notes taken by a Savioke customer satisfaction manager during regular checkup visits in February 2016. As part of the visit, the customer satisfaction manager met with one or two hotel representatives individually and asked them for (i) general feedback on using Relay, (ii) their wish list of new robot applications, and (iii) feedback on support infrastructure. Our analysis focused on the wish list notes.

5.2.2 Use Cases

We categorized the requested applications by their use cases.

People delight: All properties wanted to provide a unique experience to visitors using the robot. 3 properties specifically proposed applications designed to make their customer experience more delightful. Property A proposed an application in which the robot would approach passengers in an indoor garden area, offering them snacks or volunteering to take their picture. Properties B and C wanted the robot to roam around in their lobby and bar areas to encourage lightweight interactions with guests. Example interactions that Property B suggested included playing a game of rock-paper-scissors or sharing a joke.

Service recovery: 3 out of 5 properties wanted to use the robot to catch unsatisfied customers before they left the building. Property A requested that the robot approach passengers in the baggage claim area whenever the unloading of baggage was delayed. They wanted the robot to explain the situation and placate potentially frustrated passengers. Previously, this task was done by the airport staff, who were often not treated well by frustrated passengers. Property C wanted the robot to approach guests who were leaving the hotel, in order to ask them about their stays. Property D wanted the robot to navigate to the hotel's breakfast area and ask guests about their stays.

Mobile kiosk: 3 of 5 properties requested applications that resembled an information kiosk. Properties B and C wanted the robot to visit a couple of highly visible locations (e.g., a

location near the front entrance or the elevators), and display a series of screens encouraging interaction when people were around. They said that displaying information about the robot or the hotel would be useful, as it could trigger guests to use the delivery service or other hotel amenities in the future. Property E, which had the robot's docking station in the lobby, requested that the kiosk mode run while the robot was charging.

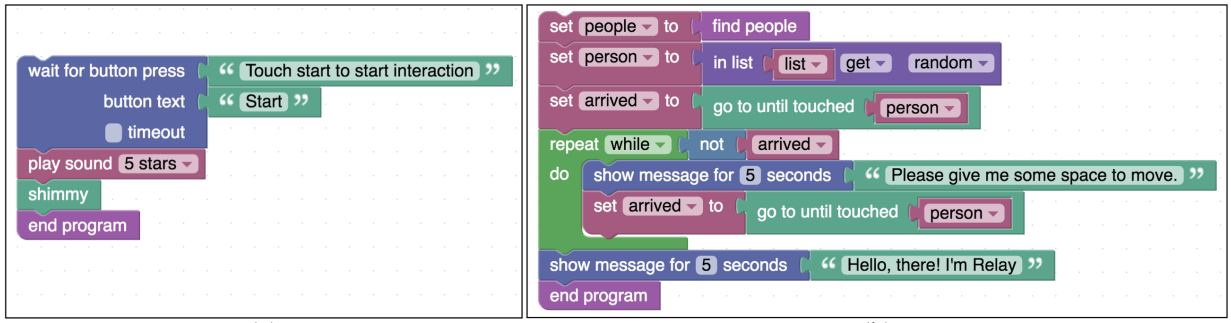
Demo: Properties B and D requested a guest-facing demo application. They often had to manually control the robot to show it in action to curious customers. Property B suggested that the application include a navigation demo and an introduction about its delivery service. Property D wanted control over how the application would be activated; they did not want guests to be able to trigger the demo, as it could interfere with actual deliveries that needed to be done.

Overall, we make the following observations:

- Having first hand experience with the Relay robot (except Property A), Savioke customers had realistic requests.
- Although their requests were similar and could be broadly categorized as above, they each had specific, custom requirements.
- Many of the requested applications involved interactions with humans.

5.3 iCustomPrograms

The software for the original room service functionality of the Relay robot was developed by Savioke's team of engineers and programmers. This team could implement many of the functionalities requested by customers (Sec. Sec. 5.2.2). However, given the diversity of requests from customers and the time it takes for custom software to be developed and deployed, this approach would not be scalable as the number of customers increase. Instead, Huang et. al developed *CustomPrograms* [93] to enable non-technical Savioke employees (e.g. marketing representatives, customer satisfaction managers, designers) as well as customers



(a) (b)

Figure 5.1: Example applications written in *iCustomPrograms*. (a) Simple interactive application; the robot first waits for a user to engage in interaction by pressing a button. It then plays a sound and shimmies in response. (b) Approaching a person application; the robot finds nearby people using **findPeople**, randomly selects a person, and approaches them. The **goToUntil** primitive returns *true* if the robot successfully reaches the destination and *false* if it is interrupted by a person tapping its touchscreen.

(e.g. hotel staff) to program the Relay robot. In this chapter, we extend *CustomPrograms* with an emphasis on *interactive* behaviors, which were a part of the applications requested by customers.

5.3.1 CustomPrograms

CustomPrograms allows users to build applications for robots by composing a set of capabilities, known as primitives, with general-purpose programming constructs like variables, loops, conditionals, and functions [93]. Applications are started manually and end when there are no more instructions to run.

Huang et al. implemented *CustomPrograms* for the Relay robot, including four categories of primitives: navigation, screen interaction, lid control, and battery state. The main navigation primitive was **goTo**, which made the robot navigate to a given location. The **shimmy** primitive was a short side-to-side swaying to convey happiness. Screen interaction primitives included displaying messages (**displayMessage**), receiving user input (e.g., **askMultipleChoice**, **askPasscode**, **askRating**), and playing non-anthropomorphic sounds

(**playSound**). The other primitives controlled the robot's lid or read the battery level.

CustomPrograms can be used to program simple interactive applications. For example, Huang et al. developed a demo application in which the robot went to several predefined locations, and offered a snack to nearby people.

5.3.2 Supporting People-Aware Behaviors

One key capability that was needed for the applications described in Sec. 5.2.2 was the ability to find and navigate to people. In *CustomPrograms*, the robot could only go to predefined locations and wait for people to interact with it. Hence, we created *iCustomPrograms*, a modified *CustomPrograms* that included a **findPeople** primitive. **findPeople** returned a list of locations where people were detected. This enabled users to create applications in which the robot approached people or recognized when someone was walking towards it. Example applications written in *iCustomPrograms* are shown in Fig. 5.1.

5.3.3 Field Study I: Airport Trials

In February 2016, we visited Property A for a two-week period. We used *iCustomPrograms* with their staff to develop two interactive applications:

Passenger Delight: In this application, the robot visited waypoints in the airport's indoor garden area. At each waypoint, it waited and approached people around it. To encourage interaction, it played a beeping sound and displayed an on-screen greeting. When a passenger started the interaction, it played a sound, displayed greeting messages, and opened its bin with snacks inside. It also did a shimmy, as an enthusiastic gesture. It then said goodbye using on-screen text and a beeping sound, and moved to the next waypoint or detected person.

Service Recovery: This was a similar application to *Passenger Delight*, but it ran in the baggage claim area whenever the unloading of baggage was delayed. Compared to *Passenger Delight* the robot was more professional. For example, we removed the enthusiastic shimmy

Figure 5.2: Pictures from the trials held at Property A. (a) On-screen text when the robot was navigating and (b) encouraging people to interact with it. (c) The robot with the Chinese New Year decal on. (d) The robot interacting with passengers.

and changed the on-screen text to politely explain the baggage situation.

We ran four trials to evaluate these applications. *Passenger Delight* was deployed for the first two trials, which took a place in the indoor garden. *Service Recovery* was deployed for the last two, which took place in the baggage claim hall. Each trial lasted 3 to 4 hours. To maximize engagement, the trials were run during Chinese New Year weekend.

For each trial, 2 to 5 airport staff members and 1 or 2 Savioke employees were present, monitoring the robot from less than 15 meters away. We tried not to interact with the passengers; however, airport staff did intervene when unexpected events happened. Examples included children acting mischievously with the robot or encountering non-English speakers.

5.3.4 Findings

We recorded observations and notes from all the meetings and trials, and conducted follow-up interviews with personnel. We identified three themes:

Problems with approaching people: The robot had difficulty approaching people naturally.

While navigating to the location of a detected person, curious crowds of people would often form around the robot, surrounding it. The robot was not programmed to recognize this situation, and continued trying to navigate to its goal location, instead of starting the interaction. This often led to people getting confused or frustrated with the robot. We resolved the issue using on-screen text asking for a clear path (Fig. 5.2), which helped the robot go through crowds.

Initiating interactions via movements and sounds: The robot initiated interactions with users in unplanned ways, e.g., just by driving around. People would follow the robot, and even tap the screen while the robot was still moving. Additionally, people noticed the robot when it played sounds.

Desire for richer control over interactive elements: The airport staff wanted the robot to have a "brighter" or "more playful" personality. They asked to have more sounds and pre-programmed movements, as well as a way to choreograph them together to make the robot look "happier." They witnessed some passengers saying "Hello" and "Goodbye" to the robot, and requested text-to-speech so the robot could respond. Finally, they also requested the ability to play background music and to format text (e.g., changing font size or adding line breaks).

5.4 Enhancements and Evaluation of iCustomPrograms

5.4.1 System Enhancements

Based on our findings, we enhanced *iCustomPrograms* as follows:

Supporting touch-to-interact: As described in Sec. 5.3.4, the robot experienced problems with people surrounding it while navigating. This was because the robot was programmed to not respond to screen input until it was done navigating. To address this, we added the **goToUntil** primitive to *iCustomPrograms*. **goToUntil** was like **goTo**, but it stopped navigating when someone touched the robot's screen. Using this made the robot behave more naturally with crowds, as they could now get the robot's attention by tapping its screen.

Fig. 5.1b is an example application illustrating the touch-to-interact behavior.

Richer control over interactive elements: We enabled users to format on-screen text in *iCustomPrograms* using HTML. We also updated the **playSound** primitive to play sounds asynchronously. In the original *CustomPrograms*, sounds were played synchronously, meaning that the robot could not navigate or respond to screen input while a sound was playing. With our change, *iCustomPrograms* supported playing long-running background music, as well as choreographing sounds with movement or on-screen interactions.

5.4.2 Improved Social Applications

We developed two new social applications using the updated *iCustomPrograms*.

People Delight was based on the *Passenger Delight* application, but was designed for use in more than just airports. It used **goToUntil** to start the interaction if a person tapped the robot's screen while it was navigating. The on-screen text was adjusted to be more property-agnostic. We also added more sounds and in-place movements to attract more attention to the robot and make the main interaction more lively.

The second application, *Mingle in Place*, was developed for smaller properties that did not want to have the robot navigating around continuously. When the application was launched, the robot navigated to a preset location and displayed three options. The first option was a demo, in which the robot described itself and its delivery service. The second option was to have the robot tell a joke. The third option was to pose for a picture, in which the robot displayed "Cheese!" on its screen. The robot played 3 to 4 different sounds during the interaction and made in-place movements. If no one interacted with the robot for over a minute, the robot attracted attention by rotating left and right while making a whistling sound. The application stopped when the battery went below a predefined threshold, or when the operator canceled it.

5.4.3 Field Study II: Trials at Five Properties

The first author demoed the *People Delight* application to Property A in February 2016 and provided a manual describing how to use it. Between March and May 2016, a Savioke customer satisfaction manager repeated the procedure with *Mingle in Place* at Properties B–E. For the hotels, room service deliveries continued to be the primary function of the robot. The properties ran *Mingle in Place* on the robot when they wanted; we did not ask them to do so for the purposes of the study.

We conducted semi-structured interviews with a staff member from Properties A, B, C, and E, after they had used the applications for at least 2 weeks. All interviewees said that the robot had successfully interacted with visitors. Property A reported that during Easter weekend, the robot was used from 10 a.m. to 6 p.m., interacting with about 500 passengers. They also said that children 7 and up, young adults, and group travelers interacted most with the robot.

Properties B, C, and E reported that their guests enjoyed interacting with the robot, especially on weekends. As with Property A, they noted that families with children and groups were most interested in interacting with it. Property E pointed out that their robot was often too busy running deliveries to use *Mingle in Place*. 3 out of 4 interviewees said that the sounds and movements of the robot helped initiate interaction with people. However, 2 out of the 4 interviewees wanted the robot to be even more interactive and have more sounds.

We recorded the number of times the Properties B–E ran *Mingle in Place*, shown in Fig. 5.3a. Due to logistical problems, we could not collect any usage measurements from Property A. Properties B and C ran *Mingle in Place* the most overall. These two properties had proposed the *People Delight* application during our formative study (Sec. 5.2.2). For most properties, the number of runs peaked in the first two weeks and gradually decreased after, which could indicate a novelty effect. However, as we heard from Property E, low usage of the application could be due to the robot being busy with room service deliveries

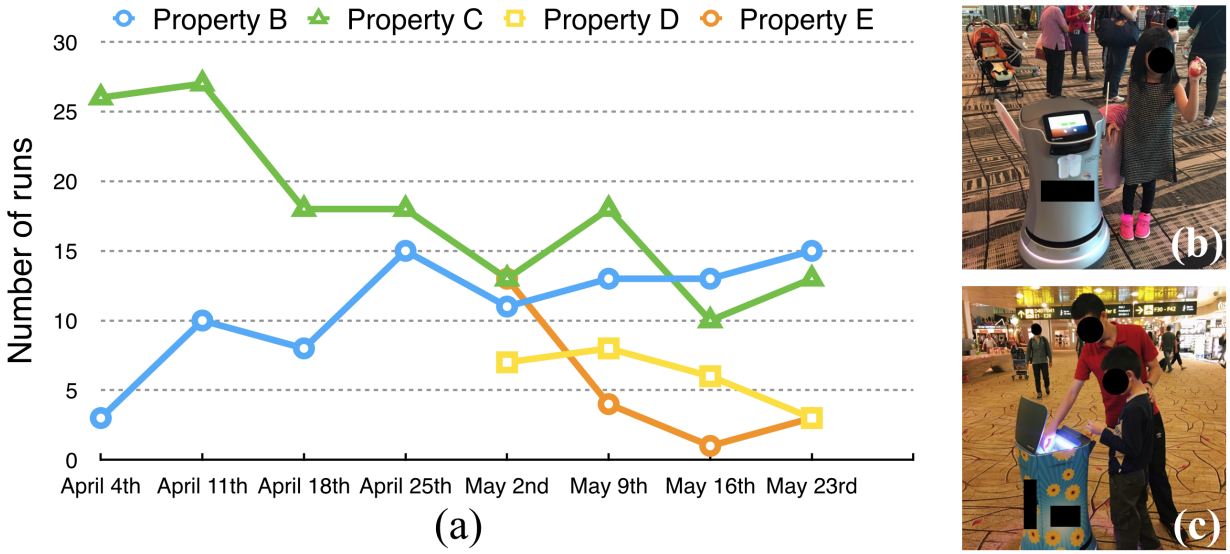


Figure 5.3: (a) Weekly *Mingle in Place* usage by property. (b) Pictures taken at Property A during the field study over Easter and (c) another local holiday weekend.

instead. And, during the study period, all of Properties B–E had run the application at least once a week.

Although we lack usage measurements from Property A, they reported that they used *People Delight* the most during Easter weekend and over another local holiday weekend in May 2016. They also said that staff members used *iCustomPrograms* to customize the contents of *People Delight* for each occasion, and applied festive decals to the robot's body (Fig. 5.3b,c).

5.5 Discussion

5.5.1 Concurrency

The updates we made to *CustomPrograms* – such as introducing the **goToUntil** primitive and changing the **playSound** primitive to be an asynchronous method – enabled users to express concurrency, which is necessary for creating interactive behaviors. However, adding or updating primitives on-demand is a not scalable approach. Hence, we believe it is impor-

tant to investigate general yet intuitive ways to support concurrency in robot programming systems, especially for those targeting non-experts. We present our investigation of supporting concurrency in a visual block-based programming system for interactive robots in Chapter 6.

5.5.2 Design implications for robot programming systems

In the future, we assume that non-programmers will want to create robot applications and behaviors to explore new use cases by themselves. Based on the experiences working with non-experts described in this chapter, we anticipated the following challenges for designing a non-expert-friendly robot programming system for socially interactive robots.

1. Enforcing design specifications. The socially interactive robot can be programmed to say or express content that is not desirable for both the users providing a service through the robot and the company that built it. For example, a user may program the robot to speak curse words, an unlikely but possible scenario. We are interested in borrowing techniques from machine learning and programming language research to allow certain users (e.g., system administrators) to define design specifications that robots must enforce, preventing unwanted behaviors (e.g., using a formal verification method).

2. Readability and reusability of end-user programs. We observed that some non-programmer users could use the *iCustomPrograms* to create robot behaviors, but their programs were not readable and hence not reusable. We anticipate that in long-term collaborative settings, the readability and reusability of end-user created programs will be significant ways to increase access to robot behavior authoring. We intend to investigate applying these proposed methods to overcome reusability issues noted here and in end-user programming literature (e.g., [224]).

5.6 Conclusion

This chapter's formative study showed that service industry workers wanted their robots to manifest socially interactive behaviors. We presented *iCustomPrograms*, a system for developing such behaviors. Robots naturally attract attention, so they must be equipped with crowd-aware navigation and interaction capabilities. In our first field study, we discovered important attributes for such interactions. In our experience, service industry workers wanted rich control over interactive elements, e.g., having more sounds, movements, and text formatting capabilities. After making these enhancements to *iCustomPrograms*, we developed and deployed social applications to five real-world service industry properties. Our target audience not only actively used the applications, but they reported interesting observations about how people interacted with the robot, information that can pave the way for future improvements in the growing numbers of socially interactive robots in the field.

Chapter 6

A COMPARISON OF CONCURRENCY INTERFACES IN BLOCK-BASED VISUAL ROBOT PROGRAMMING

6.1 Background

Concurrency makes robot programming challenging even for professional programmers, yet it is essential for rich, interactive social robot behaviors. Visual programming aims to lower the barrier for robot programming but does not support rich concurrent behavior for meaningful robotics applications. In this chapter, we present Concurrent CodeIt! (ConCodeIt!), a block-based visual programming system that enables creating different concurrent behaviors on a robot (extending our previous work on CodeIt! [93, 92, 41]). Prior work on end-user robot programming provides at least one means of expressing concurrency [127, 169, 67, 41, 48]. However, to our knowledge, there has been no work investigating and comparing alternative approaches for supporting concurrency in robot programming through simplified interfaces. We design three alternative concurrency interfaces for ConCodeIt! with (i) asynchronous procedure calls, which encourage imperative programming, (ii) callbacks, which encourage event-driven programming, and (iii) promise, which also encourages imperative programming by providing event synchronization utilities. To compare these three approaches, we conduct a systematic analysis of social robot programs with representative concurrency patterns, as well as a user study (N=23) in which participants authored such programs. Our research work in this chapter identifies characteristic differences between these different approaches and demonstrates that the promise-based concurrency interface allows for more concise and simpler programs with fewer errors.

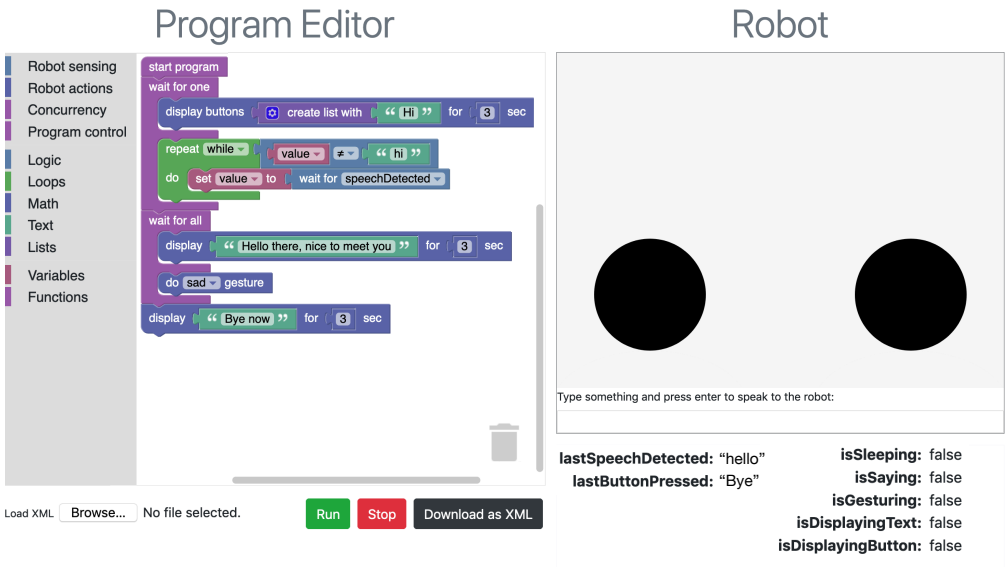


Figure 6.1: ConCodeIt! user interface.

6.2 System Overview

ConCodeIt! has three components: a general-purpose programming language with (1) custom robot action and sensing primitives and (2) custom concurrency constructs. It also includes (3) a graphical user interface.

6.2.1 Programming Language

ConCodeIt! exposes a small subset of JavaScript to users via Blockly, a block-based visual programming interface [64]. This subset includes JavaScript language constructs for expressing variables, loops, and conditionals but does not include concurrency functions such as `setTimeout`. ConCodeIt! supports concurrency with its custom functions for robot interfacing (Sec. 6.2.2) and custom concurrent constructs (Sec. 6.2.3). We choose imperative programming as the main programming paradigm for ConCodeIt! due to its pervasiveness in computing education [64, 175].

6.2.2 Robot Primitives

Robot Platform

ConCodeIt! is not tied to a particular robot, however, we focus on using ConCodeIt! for programming an idealized social robot TaRo for the purpose of this chapter. The control system for TaRo is a JavasScript library that exposes function calls for triggering robot actions, like displaying messages, and for returning (processed) sensor outputs, like the recognized speech texts of a user. We used Cycle.js to build a web application for displaying messages and buttons and the Web Speech API implementation for the Chrome browser for speech synthesis and recognition.[1] The web application was loaded on the touchscreen when the robot was turned on. For gesturing, we used the open_manipulator_controller ROS package and the roslib npm package for controlling the robot arm.[2] For the user study, we used a simulated version of TaRo, as shown in Fig. 6.1right, to conduct the study online (Sec. 6.5).

Robot Actions

Actions are robot control processes that can be started from a program and run until finished or preempted. Actions can be synchronous, i.e., block the process until the action is done, or asynchronous, i.e., start an action and move on to executing the next statement. For each action, we assume only one instance can run at a time regardless of an action's asynchrony. Available actions are shown as function definitions in Table 6.1; action names are shown in bold, arguments are in italic, and variable types are in normal font. The asynchrony of available robot actions is shown in the right column of Table 6.1.

We define the following robot actions. **say** causes the robot to say a phrase, and **gesture** makes the robot do one of a few pre-specified gestures, like a "happy" or "sad" gesture, by moving its neck and making a facial expression. Only one message can be said at a time, and only one gesture can be played at a time. **displayText** displays a text message for the

[1] `https://cycle.js.org`, `https://wicg.github.io/speech-api`

[2] `http://wiki.ros.org/open_manipulator`, `https://npmjs.com/package/roslib`

Table 6.1: Robot Actions

Name and arguments	**Asynchrony**
say(string *text*)	blocking
gesture(string *expression*)	blocking
displayText(string *text*, number *duration*)	blocking
displayButton(string[] *choices*, number *duration*)	blocking
sleep(number *duration*)	blocking
startSaying(string *text*)	blocking
startGesturing(string *expression*)	non-blocking
startDisplayText(string *text*, number *duration*)	non-blocking
startDisplayButton(string[] *choices*, number *duration*)	non-blocking
startSleeping(number *duration*)	non-blocking

user, and **displayButtons** displays a list of buttons as choices on the face screen. Only one message and only one set of buttons can be displayed at a time. **sleep** pauses program execution for the specified duration. Only one sleep can be used at a time.

Robot Events and States

Events indicate the occurrence of some change at a specific point in time. For ConCodeIt!, we defined two event categories: externally triggered and action-triggered events. Externally triggered events are initiated by some entity and sensed and processed by the robot. For example, a user finishing speaking to the robot triggers a **speechDetected** event. Action-triggered events indicate the completion of running an action, e.g., a **sayDone** event occurs when the robot finishes saying a phrase. *States* are conditions whose value can be evaluated and accessed at any given time. Events can be converted to states by storing the most recently emitted value, for example, the **lastDetectedSpeech** state stores the most recently detected speech text as a string value.

Table 6.2 shows the complete list of the robot events and states. A **buttonPressed** event is triggered when a robot detects a human pressing one of the displayed buttons

Table 6.2: Robot Events and States

Event name	**Last event state name**	**Value type**
speechDetected	**lastDetectedSpeech**	string
buttonPressed	**lastPressedButton**	string

(a) Externally triggered events

Event name	**Action status state name**	**Value type**
sayDone	**isSaying**	boolean
gestureDone	**isGesturing**	boolean
displayTextDone	**isDisplayingText**	boolean
displayButtonDone	**isDisplayingButton**	boolean
sleepDone	**isSleeping**	boolean

(b) Action-triggered events

Table 6.3: Three ConCodeIt! Interface Setups

Name	**Blocking actions**	**Non-blocking actions**	**Action events**	**Action states**	**External events**	**External states**
async	no	yes	no	yes	no	yes
callback	no	yes	yes	yes	yes	yes
waitfor	yes	no	no	yes	yes	yes

shown using **displayButtons**. The **lastPressedButton** state stores the value of the most recent **buttonPressed** event as a string. An *actionName***Done** event occurs when the robot completes an action. Finally, the **is***ActionName***ing** states indicate the running status of the robot actions.

6.2.3 Concurrent Programming in ConCodeIt!

ConCodeIt! provides the following functions to help users work with concurrent robot events and states. The `when` function takes the `eventName` string and `callback` functions as input

arguments and invokes the `callback` function when an `eventName` event occurs. The accessor functions, such as `getLast`*`EventName`* and `getIs`*`ActionName`*`ing`, return stored last event or latest state values. `wait` function takes `eventName` which blocks the process until one `eventName` event occurs, and returns the value of the occurred event. The `waitForAll` and `waitForOne` take a list of programs as input arguments; `waitForAll` blocks the process until all input programs are finished, and `waitForOne` blocks the process until one of the input programs is finished. The behavior of `waitForAll` and `waitForOne` is modeled after common promise-based synchronization utilities.[3] In fact, since ConCodeIt! is based on JavaScript, we leveraged JavaScript's features, such as event handlers and async/await promise, to implement the concurrency functions.

We present three ConCodeIt! interfaces–**async**, **callback**, and **waitfor**–that employ different combinations of the concurrency functions noted above (see Table 6.3). Each implementation supports a different concurrent programming approach. **async** is designed to provide the traditional concurrent programming experience, e.g., as in socket programming, to users. Users have access to the asynchronous function calls for starting actions and the action state accessor functions. **callback** is inspired by JavaScript's event-driven programming and *Star Wars: Building a Galaxy With Code* exercises at Code.org[4]. Similar to **async**, users can launch the robot's actions using asynchronous function calls and check the action status and latest values of externally triggered events using the state accessor functions. Users also have access to the `when` function to register a callback function; the callback function is called for occurrences of the specified robot action or external event specified by the user. **waitfor** helps to easily express concurrency behaviors while staying in the imperative programming paradigm using promise,[5] more specifically the `wait`, `waitForOne`, `waitForAll` functions that make use of promise. Like the other two implementations, users

[3] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all`, `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/race`

[4] `https://code.org/starwars`

[5] `https://en.wikipedia.org/wiki/Futures_and_promises`

have access to all state accessor functions; however, they can use blocking functions to run actions instead of using non-blocking trigger functions.

6.2.4 Graphical User Interface

The graphical user interface for ConCodeIt! uses Blockly, a library for creating block-based visual programming editors (Fig. 6.1left). Users drag and drop blocks from a toolbox of predetermined blocks onto a workspace to construct programs located on the left side of the editor. The blocks are organized into different categories – such as loops, logic, math, and ConCodeIt!-specific blocks – which can be linked through stacking or nesting or attached to variables that will be returned. Users can run or stop the program using the buttons located below the editor. Once running, the program controls the robot face displayed on the right of the interface and updates the robot status displayed below the face. Blockly enables code generation, which we use to generate the code for the ConCodeIt!-specific functions, as discussed in Sec. 6.2.2 and Sec. 6.2.3. Blockly also supports exporting a visual block program as an XML file, which we use in the "Download as XML" button displayed below the editor.

6.3 Concurrency Patterns

We are interested in the common concurrency patterns involved in robot programming. To this end, we define the concurrency patterns with (1) the source of the robot events, and (2) how users want to handle multiple events. There are two sources of the robot events, robot action and external (Sec. 6.2.2) and two ways to handle events, waiting for all events or waiting for a certain event to occur. Since there are three ways to combine the sources of events (e.g., Action-Action, Action-Event, Event-Event) and two waiting approaches, there are six concurrency patterns of interest.

Table 6.4 shows example code for three selected concurrency patterns expressed using the three ConCodeIt! implementations. The three patterns are (1) running two actions in parallel and waiting for all of them to finish, (2) waiting for one of two alternate inputs, and (3) waiting for an input event or the end of a running action. At a high-level, **async** requires

Table 6.4: Selected Concurrency Patterns in the Three ConCodeIt! Implementations

async

Action-Action & Wait-for-All
run two actions in parallel

```
startSaying("hi");
startGesturing("happy");
while (isSaying() ||
  ↪ isGesturing()) {
  sleep(1);
}
// do the next thing
```

Event-Event & Wait-for-One
wait for one of two alternate inputs

```
startDisplayingButton("Start",
  ↪ 1000)
var lastSpeechVal =
  ↪ getLastDetectedSpeech();
var lastButtonVal =
  ↪ getLastPressedButton();
while(lastSpeechVal == null
  ↪ && lastButtonVal == null)
  ↪ {
  sleep(1);
  lastSpeechVal =
    ↪ getLastDetectedSpeech();
  lastButtonVal =
    ↪ getLastPressedButton();
}
// do the next thing
```

Action-Event & Wait-ForOne
wait for input while running an action

```
startDisplayingButton("Continue",
  ↪ 1000);
var lastButtonVal =
  ↪ getLastPressedButton();
startSaying("very long
  ↪ sentence")
while(isSaying() &&
  ↪ lastButtonVal == null) {
  sleep(1);
  lastButtonVal =
    ↪ getLastPressedButton();
}
// do the next thing
```

callback

```
startSaying("hi")
startGesturing("happy")

when("sayDone", function() {
  if (isGesturing()) return;
  // do the next thing
});

when("gestureDone",
  ↪ function() {
  if (isSaying()) return;
  // do the next thing
});
```

```
startDisplayingButton("Start",
  ↪ 1000)

when("speechDetected",
  ↪ function() {
  // do the next thing
})

when("buttonPressed",
  ↪ function() {
  // do the next thing
})
```

```
displayButton("Continue",
  ↪ 1000);
startSaying("a very very long
  ↪ sentence");

when("displayingButtonDone",
  ↪ function() {
  // do the next thing
});

when("buttonPressed",
  ↪ function() {
  // do the next thing
});
```

waitfor

```
waitForAll(
  'say("Hi")',
  'gesture(happy)'
);
// do the next thing
```

```
waitForOne(
  'displayButton("Start",
    ↪ 1000)',
  'wait("speechDetected")',
  'wait("buttonPressed")'
);
// do the next thing
```

```
waitForOne(
  'displayButton("Continue",
    ↪ 1000)',
  'say("a very very long
    ↪ sentence")',
  'wait("buttonPressed")'
)
// do the next thing
```

Table 6.5: Unit Concurrent Robot Behavior Descriptions

	Wait-for-All (WA)	**Wait-for-One (WO)**
Action-Action (AA)	*Step 1:* The robot should say "Hello there!" and do the "greet" gesture. *Step 2:* The robot should say "My name is TaRo" and do the "happy" gesture.	*Step 1:* The robot should say "Hello" and sleep for 3 seconds. *Step 2:* The robot should say "timed out."
Action-Event (AE)	*Step 1:* The robot should say "Hello there!" and wait for a face to appear. *Step 2:* The robot should say "Nice to meet you!"	*Step 1:* The robot should say "Hello there, my name is TaRo. Goodbye now!" and wait for the human's face to disappear. *Step 2:* The robot should display "On standby."
Event-Event (EE)	*Step 1:* The robot should wait for the human to look to the center and stop speaking. *Step 2:* The robot should say "Hello."	*Step 1:* The robot should wait for the human to look to the left and wait for the human to look right. *Step 2:* The robot should display "Bye now!"

using a loop to check robot states; **callback** requires using the **when** functions for reacting to events; and **waitfor** requires using **waitForAll** or **waitForOne**.

6.4 Systematic Evaluation

To objectively investigate the similarities and differences among three concurrent programming approaches, we systematically evaluated the three ConCodeIt! interfaces: (1) imperative programming with asynchronous function calls, (2) event-driven programming with callbacks, and (3) imperative programming with event synchronization helper functions (Sec. 6.2.3), as well as the common concurrency pattern in robot programming (Sec. 6.3).

6.4.1 Procedure

We first designed a set of unit concurrent robot behaviors, one for each concurrency pattern described in Sec. 6.3. We describe unit behaviors in Table 6.5. We designed each behavior as a minimal robot behavior that expresses the associated concurrency pattern, and we implemented each behavior using each of the three ConCodeIt! interfaces. Where there were multiple ways to implement the desired behavior, we chose the approach requiring the least number of blocks. The resulting visual programs were stored as XML files to run our evaluation metrics.

6.4.2 Measures

We approximated the complexity of the unit behavior programs using the following measures.

Number of blocks

The total number of individual blocks used to implement a program. The three ConCodeit! interfaces have different total numbers of available block types because of the interface-specific blocks (Sec. 6.2.3). For example, the `when` block is available only in **callback**, and `waitForAll` and `waitForOne` are available only in **waitfor**.

Number of functions

The total number of function definition blocks. This measure includes the `when` block counts because it is used to define a callback function.

Number of variables

The total number of variable definition blocks.

Number of loops

The total number of loop statement blocks, such as `for` and `while` blocks.

Number of branches

The total number of if statement blocks, such as `if`, `else`, `else if`, and logical ternary blocks. Note that if an `if` statement contains multiple `else if`s, it is counted as one plus the number of `else if`s.

Number of conditions

The total number of logical operator blocks, such as negation, conjunction, and disjunction blocks, and arithmetic comparison operator blocks, such as greater than and equals.

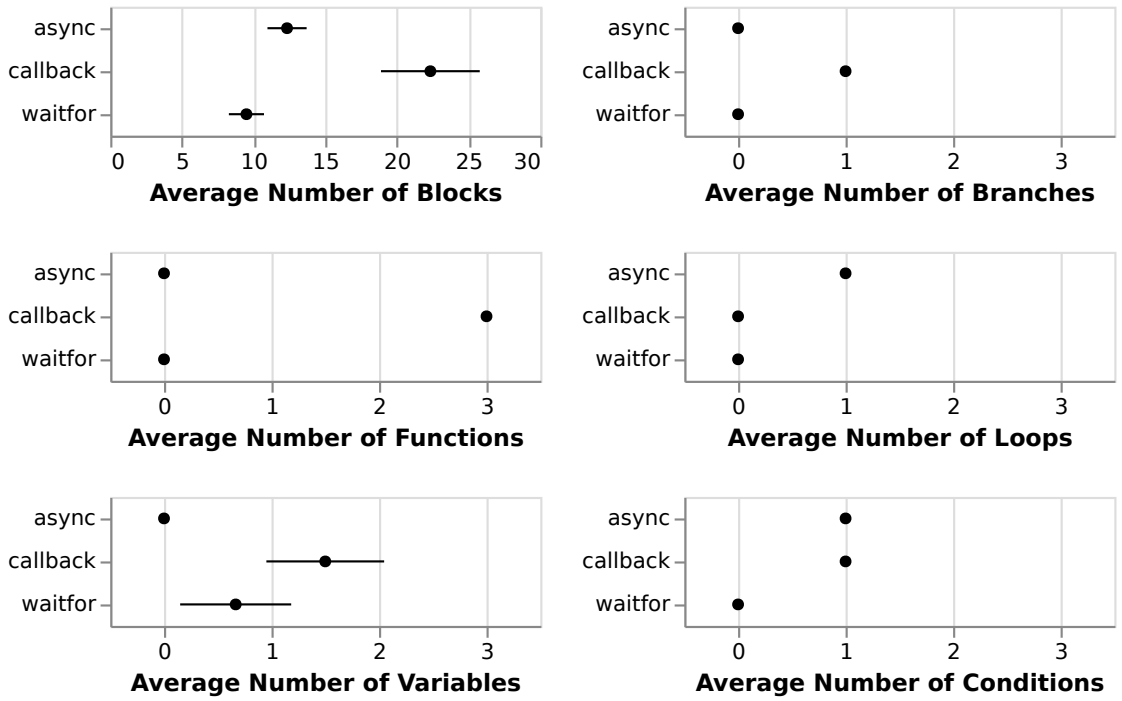


Figure 6.2: Systematic evaluation results: interfaces vs. measures.

6.4.3 Results

Fig. 6.2top shows the results. Overall, **callback** resulted in the most complex programs since it had the highest averages in most measures (5/6, i.e., all except the number of branches). Next was **async** since it had the highest averages in two measures (i.e., the number of branches and the number of conditions). Finally, **waitfor** resulted in the simplest programs since it had the lowest averages in all measures (Fig. 6.2).

Only **callback** programs required the use of functions and `if` statement blocks. These programs typically needed separate `when` blocks for each action/event to be detected, variables to monitor whether an action/event had been completed, then a separate function that was triggered in each when block to perform the final action (e.g., see Fig. 6.4b). Another reason for **callback** programs' use of the highest number of variables on average was because individual variables needed to monitor the state of the robot's interactions.

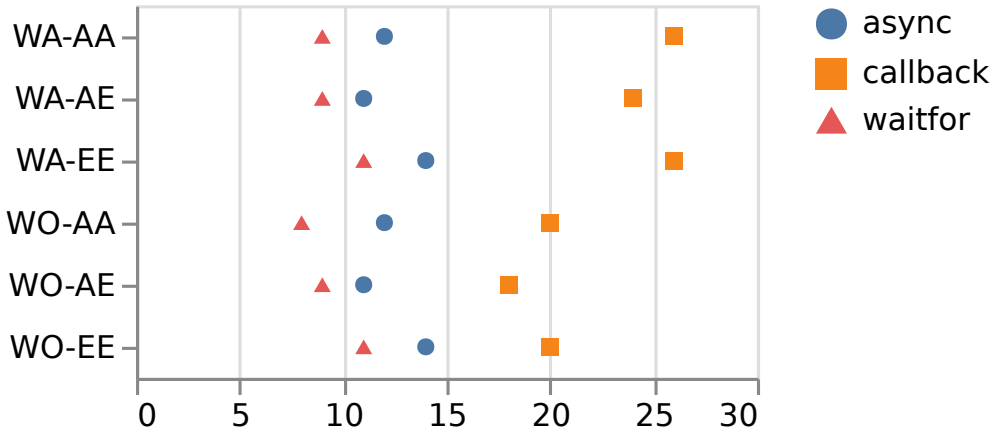
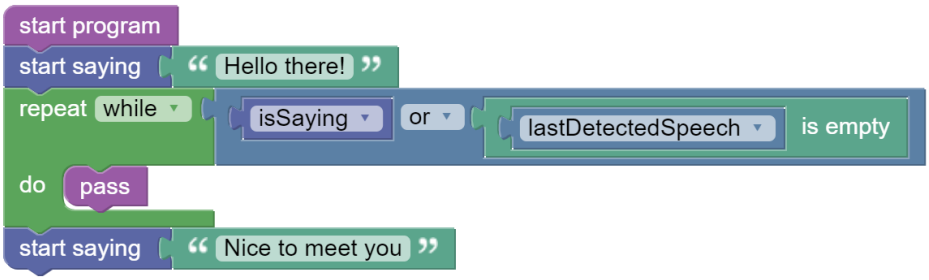


Figure 6.3: Systematic evaluation results: concurrency patterns vs. number of blocks.

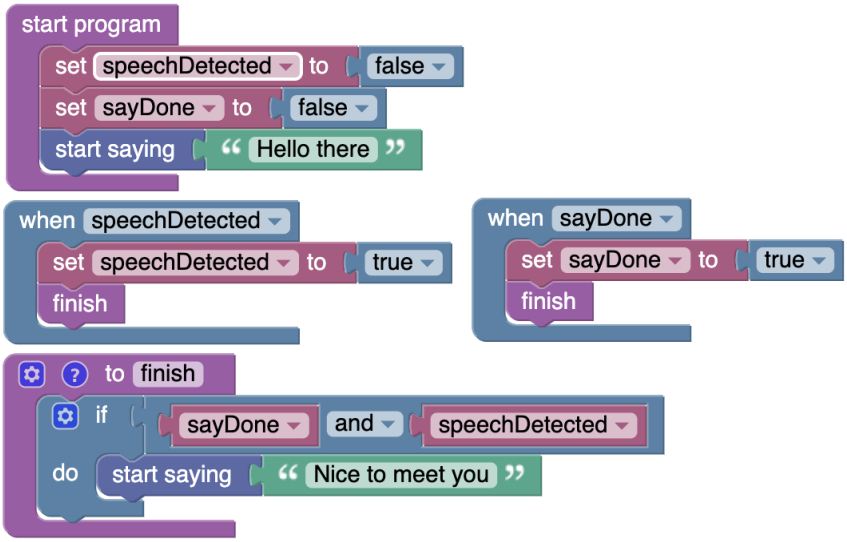
Only **async** programs required the use of a loop statement block. These programs typically needed to continuously block the rest of the program from executing until either one or all of the events/actions were completed, which was done using a while loop with a `pass` block inside (see Fig. 6.4a for an example).

Finally, only **waitfor** programs did not require the use of any logical connective or arithmetic comparison operator blocks. These programs typically used one of the concurrent blocks that waited for one or all of the actions/events within it, and no other blocks were needed (see Fig. 6.4c for an example).

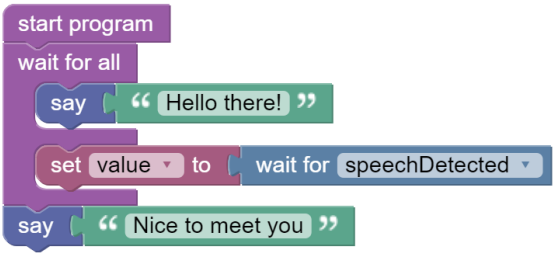
We also investigated whether a ConCodeIt! interface can implement particular concurrency patterns more concisely than others by comparing the number of total blocks required to implement six unit behavior programs over three interfaces (Fig. 6.3)bottom. We do not see significant differences across concurrency patterns within **async** or within **waitfor**. However, within **callback**, the programs implementing the wait-for-one behavior (WO) require significantly more blocks to express than the programs implementing the wait-for-all behavior (WA). This is because expressing the wait-for-one behavior in **callback** does not require additional logic for synchronizing multiple events.



(a) async



(b) callback



(c) waitfor

Figure 6.4: Wait-for-All with Action-Event (WA-AE) programs implemented using three ConCodeIt! interfaces.

6.5 User Study

6.5.1 Study Design

Like the systematic evaluation (Sec. 6.4), the user study had three conditions that represented the three concurrent robot programming approaches provided by the three ConCodeIt! interfaces (Sec. 6.2.3): **async**, **callback**, **waitfor**. The goal of the study was to compare the three concurrent programming approaches. We used a between-groups design, i.e., each participant was assigned to one of the three conditions. The study was conducted online using a Google form that contained instructions and a ConCodeIt! web page (Fig. 6.1). Participants were asked to spend around 60 to 90 minutes to complete the study and were offered a $20 Amazon.com gift card for their participation.

Participants in all conditions were shown a video tutorial explaining how to use the ConCodeIt! interface and interact with the simulated robot. In addition, all participants were given an interface-specific video tutorial to explain the assigned ConCodeIt! interface-specific functions (Sec. 6.2.3). They were then asked to implement two tasks (Table 6.6top,middle). The first was a practice task, and the solution was provided. Next, participants were asked to implement the main task and submit their program once finished. After completing the task, they answered a post-study questionnaire (Sec. 6.5.2).

6.5.2 Measures

The ease of use of each condition was estimated using the following measures.

Systematic evaluation measures

These included the six block count measures used in Sec. 6.4.2.

Correctness of programs

We developed a rubric to evaluate the correctness of participant-submitted programs (see Table 6.6bottom). To consistently measure the correctness, our rubric awarded points to

Table 6.6: User Study Task Descriptions and Rubric

Practice task
Step 1: On start, the robot should say "rain rain go away" and do the "sad" gesture. (WA-AA) *Step 2*: Once the robot finishes saying "rain rain go away" and doing a "sad" gesture, it should say "little johnny wants to play" and do the "happy" gesture. (WA-AA)
Main task
Step 1: On start, the robot should display "Press or say 'start' to begin interaction" as well as a button with "start" text. (WO-EE) *Step 2:* If the user presses the button or says "start," then the robot should introduce itself by saying "Hello, my name is TaRo" and make a happy gesture. (WO-AE) *Step 3:* If the robot finishes both actions (i.e., saying and gesturing), the robot should display a question "What is 2^{12}? You have 10 seconds to answer" as well as a button with "I'm ready to answer" text. The question and button should be displayed for 10 seconds. (WA-AA) *Step 4:* If 10 seconds pass without the user pressing the button, the robot should display "You are out of time." If the button is pressed in time, the robot should display "Please say your answer" and wait for the user to say the answer. If the user says an answer, the robot should check the answer and accordingly display "correct" or "wrong" for 5 seconds.
Rubric for the main task
Step 1.1: displays both button and text until next step *Step 1.2:* moves on when "start" button is pressed *Step 1.3:* moves on when "start" human speech is detected *Step 1.4:* correctly implements Wait-for-One behavior *Step 2.1:* introduces itself and makes a gesture *Step 2.2:* waits for "happy" gesture to finish *Step 2.3:* waits for the robot to finish speaking *Step 2.4:* only moves on when both robot actions are over *Step 3.1:* asks a question and displays a button *Step 3.2:* waits for 10 seconds *Step 3.3:* moves on when the button is pressed *Step 3.4:* displays "You are out of time" when button is not pressed after 10 second *Step 4.1:* displays text to prompt answer *Step 4.2:* waits for user to input speech *Step 4.3:* moves on when speech is detected *Step 4.4:* displays whether input is correct or not

programs based on whether each step of the instructions was correctly fulfilled or not. Each step was allocated a variable number of points, with a focus on whether the user fulfilled the concurrency requirements of a step.

Post-study questionnaire

The post-study questionnaire served as a subjective measure of ConCodeIt!'s ease of use. We asked our participants to complete the System Usability Scale (SUS) questionnaire [13]. We also asked all participants to rank their programming experience on a scale of 1 to 5, with 1 being no prior experience and 5 being a professional level of experience, and to list any programming courses they completed. The final open-ended question aimed to elicit general comments, i.e., "Any other comments about the study?"

6.5.3 Participants

The target users of ConCodeIt! are people with basic programming knowledge and no robot programming experience. We recruited participants using introductory class and general mailing lists from the University of Washington Computer Science and Engineering (CSE) Department. The participants were dividend three separate groups, one for each ConCodeIt! interface. Group 1 was asked to use **async**,Group 2 to use **callback**, and Group 3 to use **waitfor**.

There was a total of 23 participants, six for Group 1 (five female) and Group 2 (three female) conditions and nine for Group 3 (seven female). The means and standard deviations of age were: M = 21.43 & SD = 3.26 (Group 1); M = 21.71 & SD = 3.20 (Group 2); M = 19.67 & SD = 0.87 (Group 3). The means and standard deviations of self-reported programming experience were: M = 3.00 & SD = 0.58 (Group 1); M = 3.00 & SD = 1.29 (Group 2); M = 3.22 & SD = 0.67 (Group 3). Of the 23 participants, 1 of 23 had taken no programming classes, 19 of 23 completed introductory courses, 11 of 23 had experience taking mid-level programming courses, and 1 of 23 took advanced programming courses offered by the University of Washington.

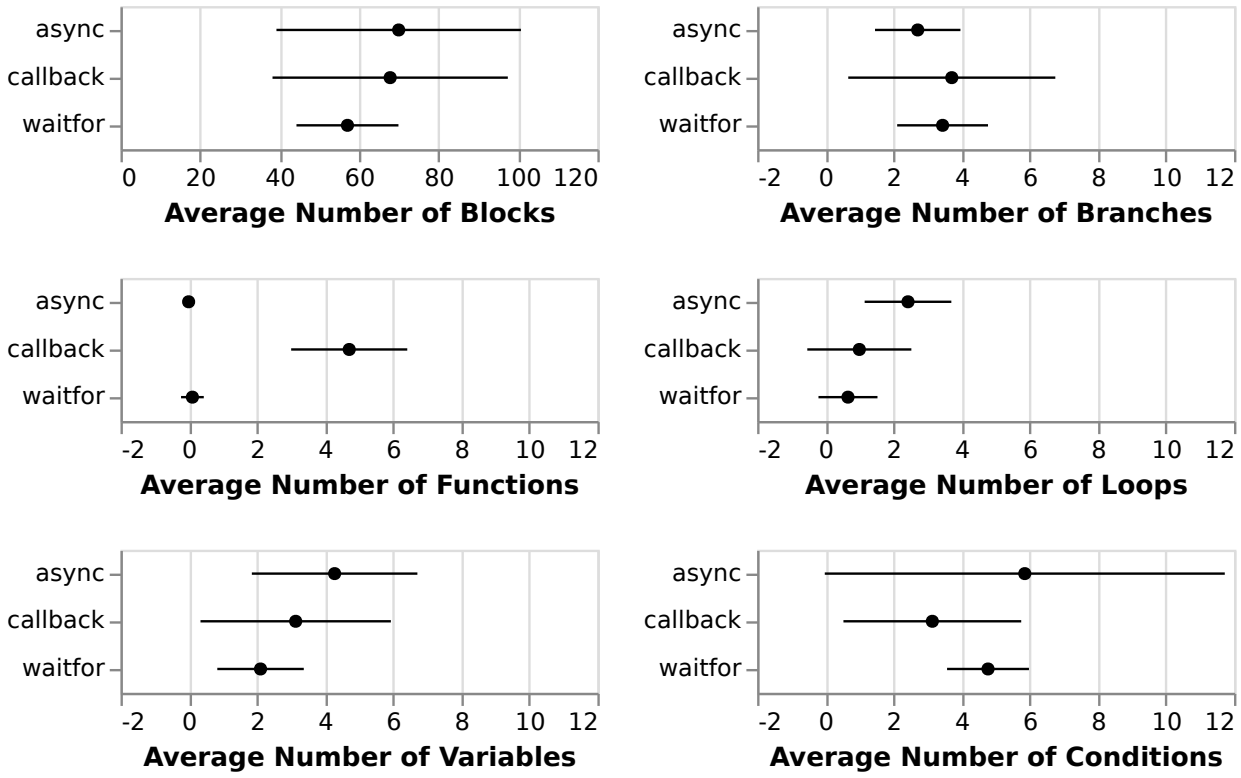


Figure 6.5: User study results: interfaces vs. measures.

6.5.4 Results

Fig. 6.5 shows the six measures applied to the participant submitted programs. Overall, the programs in **waitfor** required the lowest average number of blocks (in 3/6 measures, i.e., number of blocks, variables, and loops), and the programs in **async** required the highest average number of blocks (in 4/6 measures, i.e., number of blocks, variables, and loops, and conditions). **callback** required the highest average number of functions, and the average number of branches and **async** required the highest average number of loops.

Regarding the correctness of programs, the mean and standard deviation of score were: M = 8.14 & SD = 6.04 (Group 1); M = 8.86 & SD = 5.27 (Group 2); M = 10.56 & SD = 3.88 (Group 3). The highest possible score was 16. The relationship between the number of blocks and scores across the three interfaces is shown in Fig. 6.6left. We visualized the

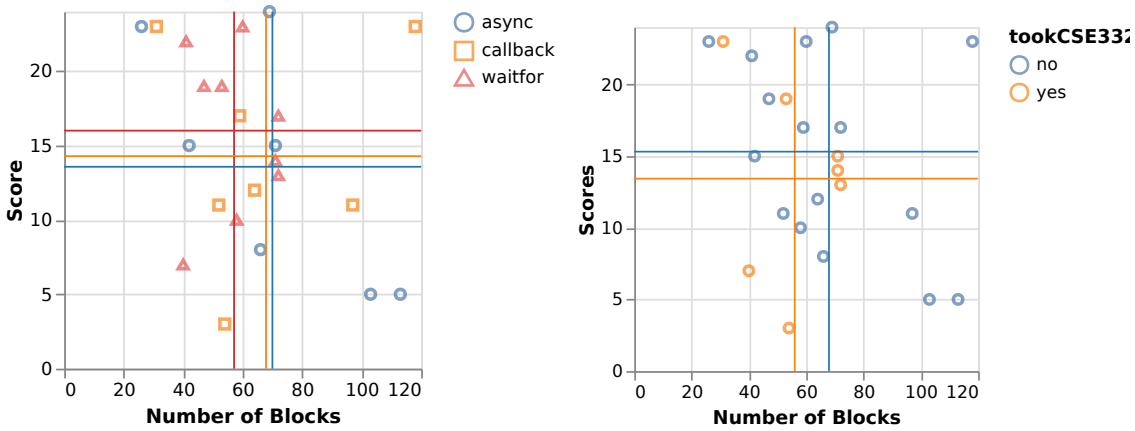
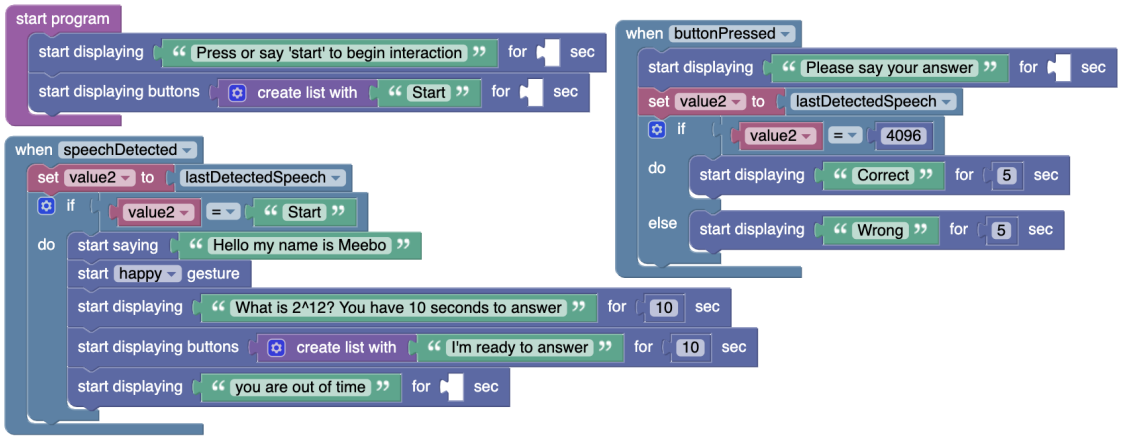


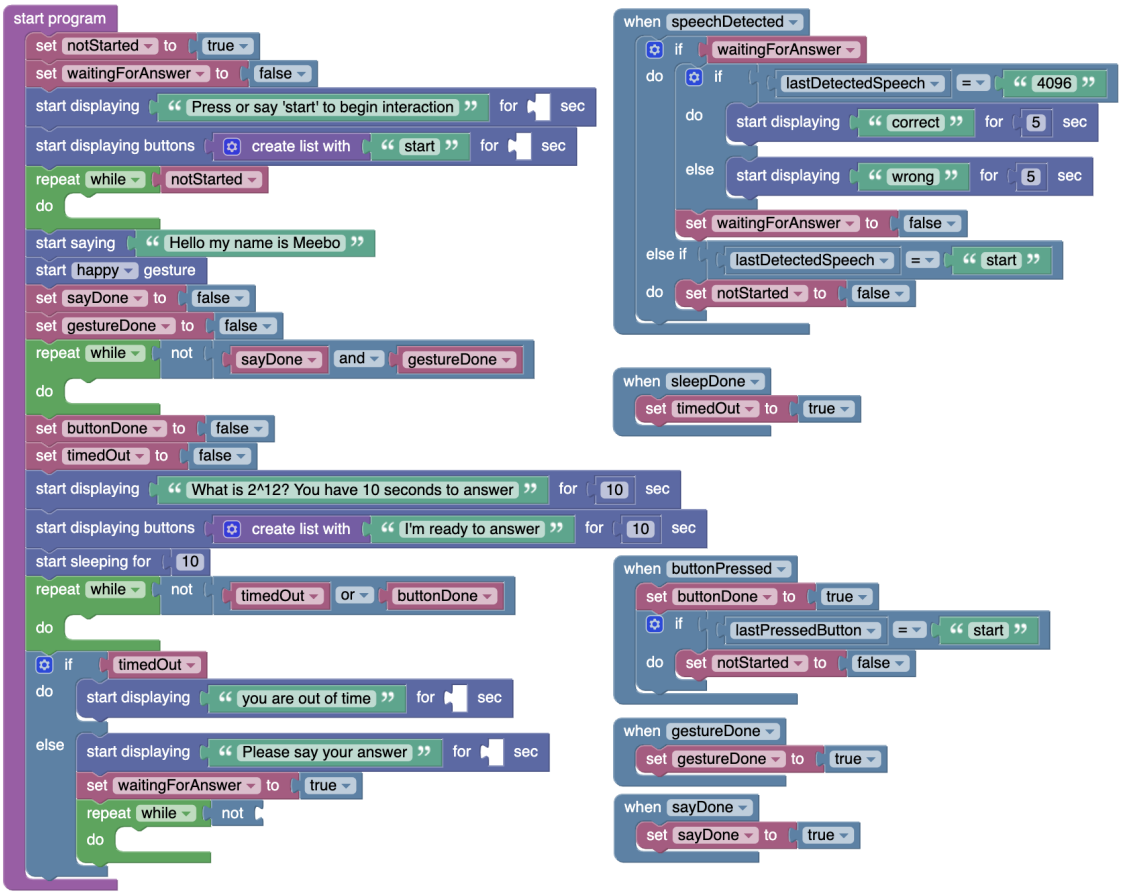
Figure 6.6: Number of blocks vs. user scores vs. interfaces (left) or vs. CSE 332 background (right). Averages are displayed as lines.

number of blocks and scores across participants who took a programming course that covers concurrency (CSE 332) and those who did not (Fig. 6.6right). The average for participants who took CSE 332 was not higher than for those who did not.

The means and standard deviations of SUS score were: M = 39.64285714 & SD = 8.469131625 (Group 1); M = 40.71428571 & SD = 20.6515882 (Group 2); M = 46.38888889 & SD = 16.63538731 (Group 3). The scores are below the SUS average score (68). Investigating the open-ended comments from 19 participants, seven mentioned issues related to the Blockly editor (*"... annoying to figure out making lists one element"*, *"Would be nice to have a zoom in/out feature on the interface."*); nine people mentioned that they did not understand how to display buttons and detect a pressed button (*"... I spent 10 minutes figuring out how to display a button"*); and three people mentioned the potential benefit of having a reference sheet (*"It would be helpful to have a glossary/element lookup."*). Only one person explicitly mentioned the challenge with concurrent programming (*"The parallel programming part definitely needs to be explained very explicitly ..."*).



(a)



(b)

Figure 6.7: User-created programs using "callback" ConCodeIt!

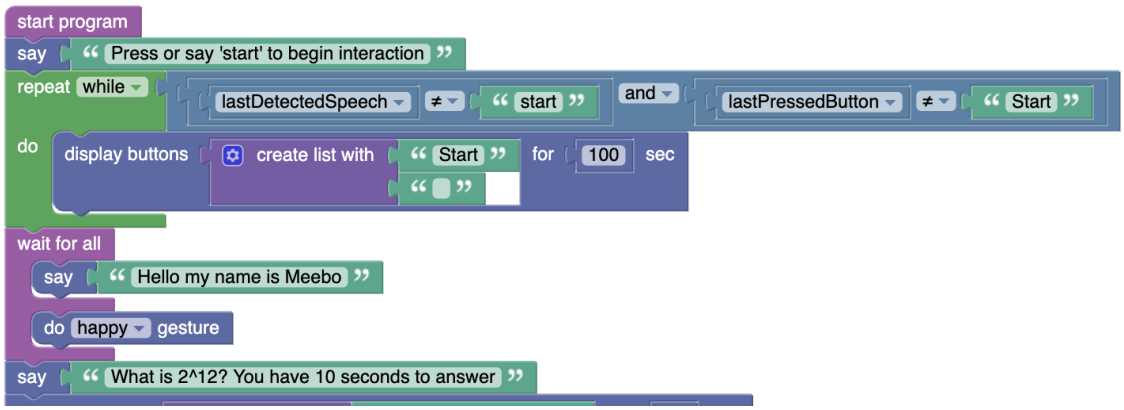


Figure 6.8: A user-created program using "waitfor" ConCodeIt!

6.5.5 User-Created Program Examples

Participants in the study thought of ways to create programs with ConCodeIt! interfaces that we did not envision when initially designing them. For example, we expected participants to use **callback** to implement the most logic in each callback function, as shown in Fig. 6.7a. However, some participants created the main loop with sub-loops for checking the global variables representing the state of the program. In this case, the callback functions (i.e., `when` blocks) were used to update the global variables to indicate the change in the state (see Fig. 6.7b). Another unexpected pattern we observed was in the programs created with **waitfor**. While **waitfor** lets programmers avoid using loops to wait for an event (see examples in Fig. 6.4a,c), some participants used loops, like the participants who used **async** did (Fig. 6.8).

6.6 Discussion

Based on our study, the **waitfor** interface, which is based on imperative programming with promise-like event synchronization utilities, was the easiest to use in the context of programming interactive robots. Participants who used this interface created the most concise programs with the highest scores on average (Sec. 6.5.4). These participants had the youngest

average age and had taken the least number of programming courses compared to participants who used the other ConCodeIt! interfaces (Sec. 6.5.3), viz., **callback** based on event-driven programming and **asynch** based on imperative programming with asynchronous procedure call utilities. However, we acknowledge that the number of participants involved in our study was small.

One question we considered throughout the study was why **callback** was challenging. Investigating the user-created programs from the user study (Sec. 6.5.5), we noticed that explicit state management (e.g., using variables to indicate which "state" the robot is in) and the multiple patterns a programmer could employ (e.g., using the main loop and global variables that are modified in callbacks vs. chaining callback functions) were the key areas of difficulty. Participants may have also had problems due to their unfamiliarity with event-driven programming. Most participants had taken the introductory and mid-level programming courses at CSE that use imperative, not event-driven, programming.

Compared to **waitfor**, the **asynch** interface, which also uses imperative programming but with asynchronous procedure call utilities, made participants perform redundant work to implement common concurrency patterns (Fig. 6.4a). Overall, we believe that adding minimal features to support concurrency while allowing programmers to keep a consistent mental model is key to achieving ease of use while supporting desired concurrency behaviors.

6.7 Conclusion

This chapter presented ConCodeIt!, a block-based visual programming system for programming interactive robots. We first defined a framework for (1) identifying programming constructs required for expressing concurrency, and (2) categorizing common concurrency patterns in the context of programming robots. We then proposed three programming systems that represent common approaches for expressing concurrency and compared them via a systematic evaluation and an online user study. Our results show that the imperative programming paradigm with synchronization support produced more concise and predictable programs, while the event-driven one was more challenging for robot programmers.

Chapter 7

ITERATIVE REPAIR OF SOCIAL ROBOT PROGRAMS FROM IMPLICIT USER FEEDBACK

7.1 Background

Creating natural, autonomous interactions with social robots requires rich, multi-modal sensory input from the user. In addition to the challenges associated with concurrency (Chapter 6), writing interactive robot programs that make use of such input often involves tedious and error-prone tuning of program parameters, such as thresholds on noisy sensory streams for considering the user disengaged from the interaction. This tuning process dealing with low-level streams and parameters makes programming of social robots time-consuming and inaccessible for people who could benefit the most from unique use cases of social robots. To address this challenge we propose to use iterative program repair, whereby the programmer creates an initial program sketch in Social Robot Program Transition Sketch Language (SoRTSketch)–a domain specific language that supports expressing uncertainties related to thresholds in transition functions. The program is then iteratively repaired using Bayesian inference based on annotations of the program's interaction traces that are either provided by the programmer or inferred from implicit feedback given by the user during the interaction. We demonstrate the effectiveness of this approach in improving social robot programs that represent three common human-robot interaction patterns, first with a human simulator and then with 10 human users. We also show how our approach helps programs adapt to environment changes over time.

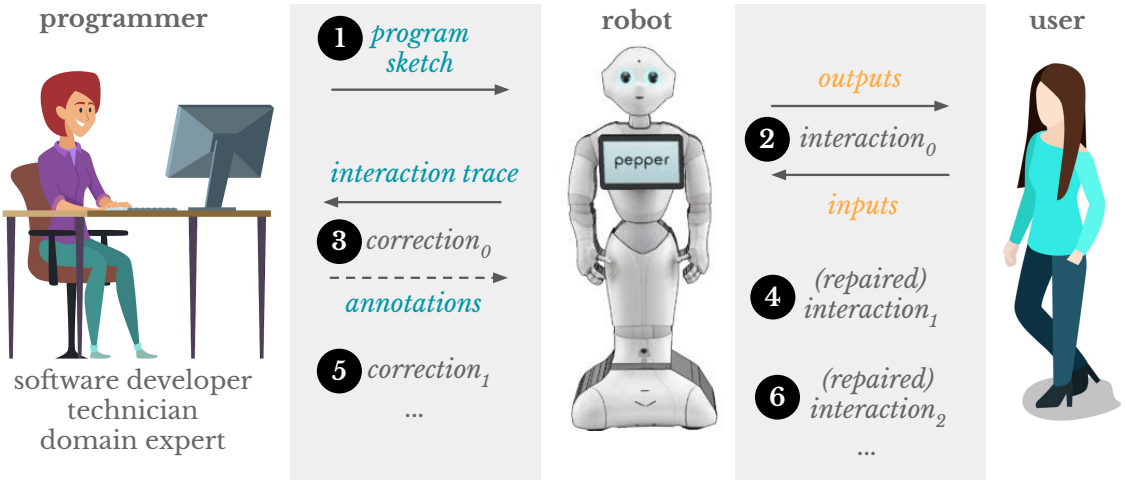


Figure 7.1: Overview of the iterative program repair process. (1) The programmer creates a *program sketch* representing a finite state machine, or FSM, (2) the program is executed on the robot, and a user interacts with the robot, (3) the program is automatically repaired based on annotations over the interaction trace that are either provided by the programmer or derived from implicit feedback in user inputs, (4) steps 2-3 are repeated until a satisfactory program is obtained.

7.2 Approach

Our approach, summarized in Fig. 7.1, starts with a programmer creating a program sketch that is iteratively refined through interactions with a user. We now describe this process in detail.

7.2.1 Program Sketches

A *program sketch* is a partial program that encodes the high-level structure of a solution while leaving low-level details unspecified [194]. Program details of a program can be left unspecified using hole variables that are later derived by the repair algorithm. In this chapter, we present **SoRTSketch** (Social Robot Program Transition Sketch Language), a domain-specific language for sketching social robot behaviors based on finite state machines (FSMs) with transitions that are not fully specified.

FSM description

SoRTSketch is based on FSMs, widely used to represent robot behaviors [29, 169, 20, 149, 151, 133, 187, 30, 4]. Specifically, we represent social robot behaviors as discrete-time *Mealy machines* with continuous inputs and outputs and program variables. Formally, our FSM is a tuple $(S, S_0, V, V_0, \Sigma, \Lambda, T)$, where S is a finite set of states, S_0 is the start state, V is a finite set of program variable values, V_0 is the start program variable values, $\Sigma \in \mathbb{R}^n$ are continuous inputs (obtained from robot sensors), $\Lambda \in \mathbb{R}^m$ are continuous outputs (executed as robot actions), and $T : S \times V \times \Sigma \rightarrow S \times V \times \Lambda$ is a transition function. At each time step t, the transition function is executed to update the state $s_{t+1} \in S$ and program variables $v_{t+1} \in V$ (an update could be $s_{t+1} = s_t$ or $v_{t+1} = v_t$) and to output a robot action to be triggered in next time step $\lambda_{t+1} \in \Lambda$ (could be *noaction*).

Language description

A transition function in SoRTSketch consists of an `if-else` statement that checks the values of variables derived from the current FSM state $s \in S$, variables $v_t \in V$, and sensor inputs $(x_t^1, ..x_t^n) \in \Sigma$. It assigns values of the new FSM state $s_{t+1} \in S$, variables $v_{t+1} \in V$, and output action $a_{t+1} \in \Lambda$, accordingly. Sketching capabilities leverage the fact that the condition expressions in `if` statements can include *hole* variables instead of constants. Fig. 7.2 shows part of the formal syntax for representing transition functions in SoRTSketch.

As an example, consider an interactive storytelling social robot. To ensure the user pays attention while the story unfolds, the robot can move into a paused `"wait"` state whenever it detects that the user has disengaged. This requires the transition function to encode *disengagement* in the `if` expression, by comparing an input variable (such as the person's gaze direction `faceYawAngle`) with a threshold `maxEngagedAngle`. Where conventional programs would require assigning `maxEngagedAngle` to a constant, SoRTSketch lets the variable be added to a list of hole variables for later repair.

Programmers can create a hole variable whenever they are uncertain about the exact

⟨*uniary-operator*⟩ ::= '`-`' | '`!`' | '`abs`' | ...

⟨*binary-operator*⟩ ::= '`+`' | '`*`' | '`>`' | '`<`' | '`&&`' | '`||`' | '`==`' | ...

⟨*temporal-operator*⟩ ::= '`delay`' | '`debounce`' | '`average`' | ...

⟨*expression*⟩ ::= k *constant*
| s *state*
| v *variable*
| x *input*
| θ ***hole***
| ⟨*uniary-operator*⟩ ⟨*expression*⟩
| ⟨*expression*⟩ ⟨*binary-operator*⟩ ⟨*expression*⟩
| ⟨*expression*⟩ '`?`' ⟨*expression*⟩ '`:`' ⟨*expression*⟩ *ternary operator*
| ⟨*temporal-operator*⟩ '`(`' ⟨*expression*⟩ '`,`' ⟨*expression*⟩ '`)`'

⟨*statement*⟩ ::= '`return`' s;
| '`if (`' ⟨*expression*⟩ '`) {`' ⟨*statement*⟩ '`} else {`' ⟨*statement*⟩ '`}`'
| ⟨*statement*⟩

⟨*transition*⟩ ::= ⟨*statement*⟩

Figure 7.2: Formal syntax of SoRTSketch for representing FSM transition functions as program sketches.

value of a constant needed for the transition function. Further, they must also specify the distribution of the variable as a probability density function (e.g., Bernoulli). For example, the programmer of the transition program in Fig. 7.3bottom used hole variables for perceptual thresholds (e.g., `maxEngagedAngle`), timing thresholds (e.g., `engagedTimeout`), and operator selectors (e.g., `useAvgOp`). As described in Sec. 7.2.3, this lets the repair algorithm treat hole variables as random variables in Bayesian inference.

Temporal operators

Due to noise in a robot's sensory inputs, knowing only the latest value of sensory inputs can be insufficient for creating fluent human-robot interactions. Therefore, SoRTSketch includes three temporal operators that have access to the history of a sensory stream:

1. `average`(x, τ) returns the average value of an input variable x over the specified duration τ into the past; i.e., between $t - \tau$ and t.

2. `delay`(x, τ) returns the value of an input variable x from duration τ into the past; i.e., it creates an input stream delayed by τ.

3. `debounce`$(\textit{expression}, \tau)$ returns the conjunction of the boolean *expression* involving an input variable, over the specified duration τ into the past.

As an example, if the `faceYawAngle` input variable is known to be noisy, an `if` expression could compare `average(faceYawAngle, historyDuration)` to a threshold `maxAverageEngagedAngle`, where both duration and threshold variables could be holes.

Example program

Fig. 7.3bottom shows instantiations of different categories of expressions with holes in a transition function implemented with SoRTSketch for the storytelling robot behavior. For example, it shows: how holes can represent uncertainty in deciding which operator to use

(`useAvgOp`); the use of multiple sensory streams (`faceYawAngle` and `voiceLevel`) over time; within a complex expression with multiple parameters, to represent high-level concepts like engagement; and how to make decisions about state transitions. For expository reasons, we simplified the transition function program to only return the next state and displayed the minimal subset of the DSL syntax required for implementing the transition function in Fig. 7.2 and hide certain FSM components such as variables and outputs in the FSM graph visualization (Fig. 7.3top).

7.2.2 State Traces and Corrections

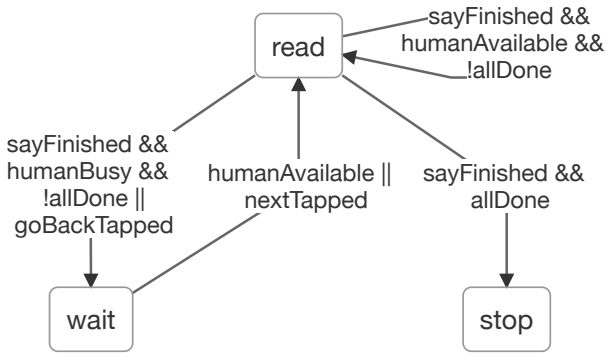
Program execution

SoRTSketch program sketches are executable even before they are repaired. For example, the holes can be set to the sampled values from the given distributions. When SoRTSketch programs are executed on a robot, users can interact with the robot in different ways. We refer to a sequence of inputs $I = [(x_1^1, ...x_1^n), ..., (x_\mathcal{T}^1, ...x_\mathcal{T}^n)]$ received over $\mathcal{T}$ steps as an *input trace* and the state values resulting from running the FSM, $O = [s_1, ..., s_\mathcal{T}]$ as a *state trace*. If the program's hole variables are assigned to a set of "good" values, i.e., ones that make the robot behave exactly as the programmer intended during the interaction, the state trace for that interaction can be considered correct.

However, interactions with a program before hole variables are adjusted through program repair often involve errors. We can measure the correctness of program execution based on the *overlap* between the state trace of that interaction and the state trace of a program that behaves exactly as the programmer intends. Given a state trace, the programmer can annotate the incorrect states and correct the robot's state.

Implicit state correction

Correcting state traces of user interactions requires programmers to be in-the-loop after deployment. While feasible, this might be redundant in some cases where the interaction



Category	Variable name	Example value
constant	`nSentences`	10
state	`curState`	`"read"`
variable	`curSentenceIndex`, `hold`	1, `false`
input	`faceYawAngle`, `voiceLevel`, `sayFinished`, `goBackTapped`, `nextTapped`	7, 0.1, 0, 0, 0
hole	`minDisengagedAngle`, `disengagedTimeout`, `maxEngagedAngle`, `engagedTimeout`, `speakingWindow`, `minSpeakingLevel`, `useAvgOp`	15, 1500, 30, 500, 1000, 0.4, `true`

```
1   lookingAtRobot = debounce(abs(faceYawAngle < minDisengagedAngle), disengagedTimeout);
2   lookingAway = debounce(abs(faceYawAngle > maxEngagedAngle), engagedTimeout);
3   speaking = useAvgOp ? average(voiceLevel, speakingWindow) > minSpeakingLevel : debounce(voiceLevel,
    ↪  speakingWindow) > minSpeakingLevel;
4   allDone = sayFinished && (curSentenceIndex == nSentences)
5   humanAvailable = !lookingAtRobot && !speaking;
6   humanBusy = lookingAway || speaking;

7   if (curState == "read" && sayFinished && humanAvailable && !allDone) {
8     return "read";
9   } else if (curState == "read" && (sayFinished && humanBusy && !allDone || goBackTapped)) {
10    return "wait";
11  } else if (curState == "read" && sayFinished && allDone) {
12    return "stop";
13  } else if (curState == "wait" && (humanAvailable || nextTapped && !hold))) {
14    return "read";
15  }
```

Figure 7.3: An example program in SoRTSketch. (top) the social robot used this paper and an FSM visualization for the storytelling program; (middle) Instantiation of SoRTSketch in a storytelling social robot domain with defined constants, inputs, variables, and holes; (bottom) transition function program with variable definitions and `if-else` statements.

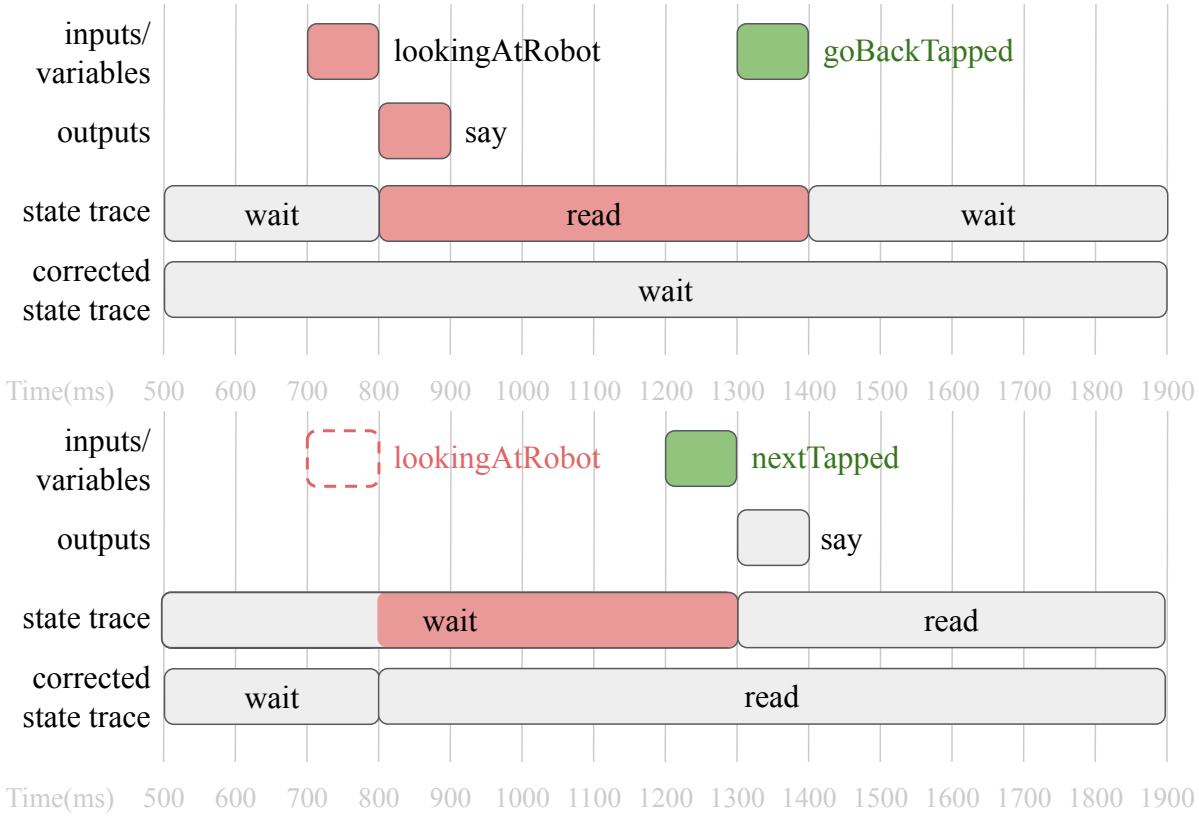


Figure 7.4: Incorrect (above) and missing (below) transition errors and recovery examples from the storytelling robot example. Incorrect states and associated variables are highlighted in red. Implicit feedback in the user input is highlighted in green. The dotted red square represents an undetected human action.

trace itself contains information about errors that occur. In particular, two types of transition errors that occur during FSM executions can cause errors in state traces.

1. *Incorrect transition*, which occur when the robot makes a transition to a different state when it was supposed to remain in the same one.

2. *Missed transition*, which occur when the robot remains in the same state when it was supposed to transition to a different one.

For instance, the robot in the storytelling example might make an incorrect transition from the `"wait"` state to the `"read"` state if `minSpeakingLevel` or `speakingWindow` is set too low, i.e., the robot mistakenly assumes the user finished speaking. Similarly, a missed transition may occur if `minDisengagedAngle` threshold is not set properly, e.g., the robot remains in `"wait"` state even after the user is ready to listen. These two types of transition errors can derail an interaction if no way to recover. Since such errors cannot be completely avoided, it is important for human-robot interactions to give users a way to recover from errors. An example of a recovery mechanism is the use of "Go back" and "Next" buttons (or other input commands, such as speech or gesture) that let the user to guide the interaction when faced with incorrect and missed transitions.

These mechanisms for repairing the interaction can also serve as implicit feedback for repairing the program; the use of interaction repair mechanisms in the state trace can be converted to corrected state traces usable by the repair algorithm. Alg. 1 shows how this is achieved. A user tapping "Go back" button indicates that the previous transition was incorrect; hence, all states between the last transition and the button tapping in the trace should be annotated as the previous state value. A user tapping the "Next" button indicates a missed transition; in this case, all past states within a predefined window size $\mathcal{W}$ should be annotated as the next state value.

Algorithm 1 ImplicitStateCorrection

Input: State trace $\mathbf{O} = [s_1, ..., s_\mathcal{T}]$ and window size $\mathcal{W}$
Output: Corrected state trace $\mathbf{O}'$

Initialize j to 1
for $i = 2, ..., \mathcal{T}$ **do**
  if $s_i \neq s_{i-1}$ **then**
    if $GoBack(s_{i-1}) = s_i$ **then**
      $s_j \leftarrow s_i, ...s_{i-1} \leftarrow s_i$
    else if $Next(s_{i-1}) = s_i$ **then**
      $s_{\max(i-\mathcal{W},1)} \leftarrow s_i, ...s_{i-1} \leftarrow s_i$
    end if
    $j \leftarrow i - 1$
  end if
end for
$\mathbf{O}' = [s_1, ..., s_\mathcal{T}]$

7.2.3 Iterative Program Repair

Given a program sketch and corrections over program interaction traces, program repair aims to find the best set of program parameters represented by hole variables in the sketch. We present the first algorithm that achieves this by searching over all combinations of parameter value assignment combinations, as summarized in Alg. 2. Hole variables Θ_1 are initialized based on the variable distribution the programmer provides. The program is then executed to obtain an input trace I_1 and corrected state trace $\widehat{O}_1$. The main update step involves evaluating the following equation:

$$Repair(K, \mathbf{I}, \mathbf{O}) = \underset{\Theta}{\operatorname{argmax}} \sum_{j=1}^{|\mathbf{I}|} Overlap(K[H := \Theta](I_j), \widehat{O}_j).$$

Given an initial program sketch K and all program input traces $\mathbf{I} = (I_1, ..., I_i)$ and corrected state traces $\mathbf{O} = (\widehat{O}_1, ..., \widehat{O}_i)$ up to the current iteration i, the goal is to find the hole variable values that maximize the total overlap between corrected state traces $\mathbf{O}$ and output state traces from executing the program sketch with those hole variable values $K[H := \Theta]$.

Algorithm 2 IterativeRepair

Input: Program sketch K and initial values for holes Θ_1

initialize $\mathbf{I}$ and $\mathbf{O}$ to empty sets
for $i = 1, 2, ...$ **do**
    run the program $K[H := \Theta_i]$ and record program inputs I_i and output state trace O_i
    $\widehat{O}_i \leftarrow$ ImplicitStateCorrection(O_i)
    add I_i and $\widehat{O}_i$ into $\mathbf{I}$ and $\mathbf{O}$, respectively
    $\Theta_{i+1} \leftarrow \mathit{Repair}(K, \mathbf{I}, \mathbf{O})$
end for

Algorithm 3 IterativeBayesRepair

Input: Program sketch K and prior $\Pr(\Theta_1)$

for $i = 1, 2, ...$ **do**
    run the program $K[H := \Theta_i]$ and record program inputs I_i and output state trace O_i
    $\widehat{O}_i \leftarrow$ ImplicitStateCorrection(O_i)
    $\Theta_i \leftarrow \mathit{BayesRepair}(K, I_i, \widehat{O}_i)$
    $\Pr(\Theta_{i+1}) \leftarrow \Pr(\Theta_i | \widehat{O}_i; K, I_i)$
end for

Intuitively, the algorithm searches for the hole values that make the program as consistent as possible with the corrected state traces.

This approach requires saving all traces of a program, which is not memory-efficient, and solving a harder satisfiability problem as increasing numbers of traces are obtained. Also, the user's behavior or the operation environment may change over time, in which case using all past traces could degrade performance. To address these issues, we propose an alternative algorithm, Alg. 3, that uses Bayesian filtering, a technique commonly applied to analyze sequential data. The main update step involves evaluating the following equation:

$$\mathit{BayesRepair}(K, I_i, \widehat{O}_i) = \underset{\Theta_i}{\operatorname{argmax}} \ \Pr(\Theta_i | \widehat{O}_i; K, I_i), \tag{7.1}$$

where:

$$\Pr(\Theta_i | \widehat{O}_i; K, I_i) = \frac{\Pr(\widehat{O}_i | \Theta_i; K, I_i) \Pr(\Theta_i)}{\sum_{j=1}^{|\mathbf{O}|} \Pr(\widehat{O}_j | \Theta_j; K, I_j) \Pr(\Theta_j)} \tag{7.2}$$

and the likelihood function is a percentage overlap function

$$\Pr(\widehat{O}_i|\Theta_i; K, I_i) \propto Overlap(K[H := \Theta_i](I_i), \widehat{O}_i). \tag{7.3}$$

At every iteration i, the algorithm first computes the posterior (7.2) and then uses a maximum a posteriori (MAP) estimation to determine hole variable values (7.1). The subsequent repair starts by setting the new prior to the current posterior using

$$\Pr(\Theta_{i+1}) \leftarrow \Pr(\Theta_i|\widehat{O}_i; K, I_i).$$

At a high level, IterativeBayesRepair compactly encodes information from the past into the prior $\Pr(\Theta_i)$ instead of computing it from stored data at every iteration. For the repair algorithms in both approaches, we use an exhaustive enumeration algorithm by binning parameter spaces into discrete sets.

We built the entire system in JavaScript. We used Cycle.js framework [197] to implement robot behaviors as Cycle.js applications and tools for recording and replaying interaction traces. The repair algorithms and ImplicitStateCorrection were implemented from scratch.

7.3 Evaluation

To evaluate the feasibility and effectiveness of our approach, we conducted: (1) a simulation experiment using stochastic human simulators, (2) a human experiment with 10 participants and a tabletop robot.

7.3.1 Social Robot Tasks

We evaluated the repair of three human-robot interaction programs. Each represents an interaction pattern that is common across social robot application domains [176, 37, 6, 22, 184, 68], as described below:

Storytelling

The robot reads a story line-by-line until the story ends. The robot should pause the reading when the user looks away or interrupt the robot by speaking to it. The robot should resume the reading when the user looks at the robot and no longer speaks. During the interaction, the robot offers "Pause" and "Resume" buttons to let the user to manually control it in case it incorrectly pauses or resumes the reading. The storytelling scenario captures how robots handle *engagement and disengagement patterns* [22, 68] that is applicable to the service robots such as an information booth robot [22]. Note that the storytelling FSM used in this section is different from the one used in Sec. 7.2.

Neck Exercise

The robot instructs the user to perform a sequence of neck exercises. It should display "Tilt your head to the LEFT" and move on to the next instruction when it detects the user has correctly followed the first one. The robot repeats the two different instructions three times (i.e., six instructions in total). During the interaction, the robot offers the "Next" and "Go back" buttons to let the user manually switch to the next or previous instruction in case the robot does not detect or incorrectly detects the user's action. The neck exercise represents *instruction and monitor patterns* [184] that require the robot to detect the human action, either to verify completion of an instructed task or as an input mechanism.

Open-Ended Q&A

The robot asks a series of open-ended questions. It displays each question on the screen and waits for the user's verbal response. While waiting, the robot should be aware of the user's use of gaze-aversion for holding the conversation floor [6]. When the robot detects the user has finished answering the question, it should ask the next one, for a total of five questions. During the interaction, the robot offers the "Next" and "Go back" buttons to let users manually switch to the next or previous question in case it mistakenly does or does

not move on correctly. Open-ended Q&As represent the *turn-taking pattern*, which requires the robot to detect multi-model turn-yielding signals [176, 37, 6].

Each interaction program has a transition function that depends on features computed from continuous sensor streams of the robot. We repaired 5 (storytelling), 3 (neck exercise), and 7 (Q&A) FSM transition parameters across 2, 2, and 1 transitions, respectively.

7.3.2 Simulation Experiment

To systemically evaluate our approach with larger amounts of data, we created a human simulator to obtain FSM input traces and ground truth state traces for each interaction. The human simulators started with an initial human intention (e.g., engage or disengage), which changed its value after a time interval sampled from a predefined uniform distribution per interaction. We created ground truth state traces by sampling values from the simulated intention traces at the robot sensor streaming frequency (10hz). FSM input traces were created by simulating sensor input values using the uniform distributions defined per each intention and interaction. To emulate an experiment involving real human users, we selected simulator parameters based on data collected from author interactions with the robot for each interaction. For the storytelling, neck exercise, and open-ended Q&A interactions, the human simulator stopped when it changed the ground truth states 5, 5, 6, respectively, and the average total duration of 100 simulated FSM input traces was 27.02 (SD = 5.53), 52.55 (SD = 5.05), 61.63 (SD = 6.78) seconds, respectively.

Procedure

We first simulated 100 input traces and ground truth state traces as a test dataset. At each iteration, we simulated a new FSM input trace and desired state trace pair and then applied the IterativeRepair (Alg. 2) and IterativeBayesRepair (Alg. 3) to acquire a set of repaired parameters. For the IterativeBayesRepair, we compared using the ground truth state trace to using the noisy state trace for repair, where the latter represented noisy corrections provided by the real programmer or user. The noisy state trace was computed by adding uniform

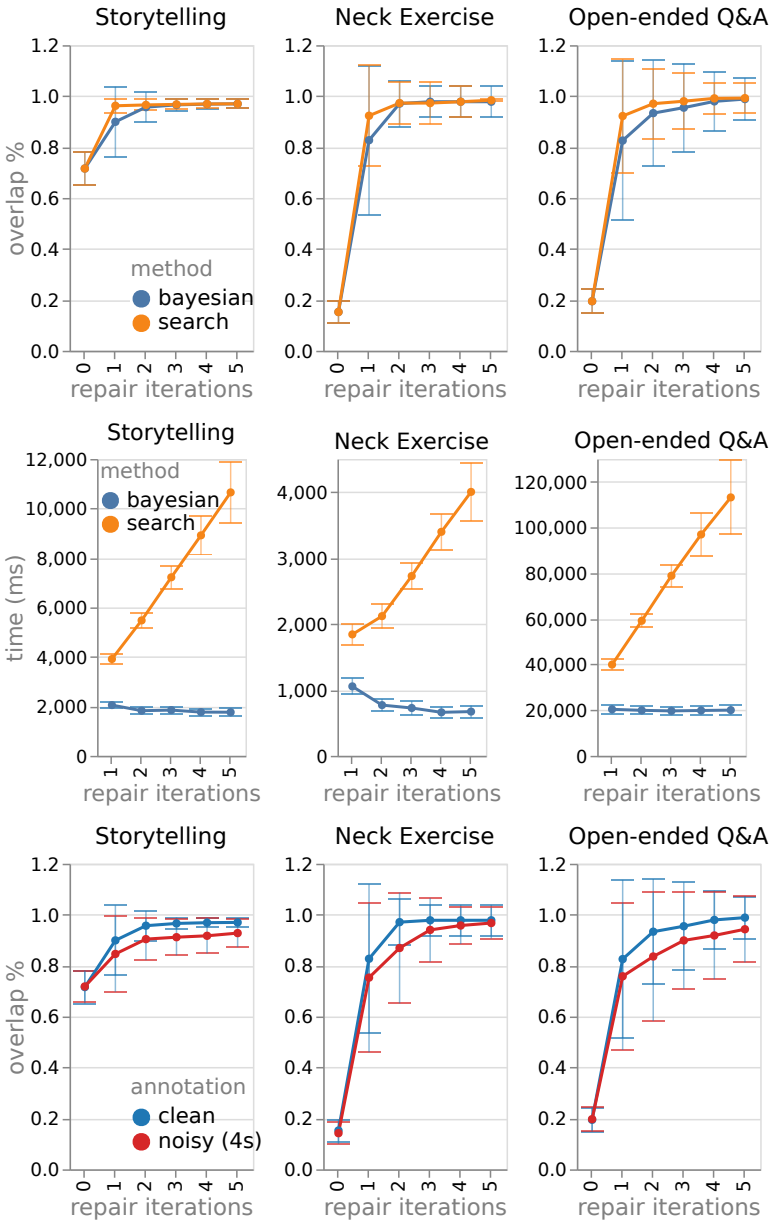


Figure 7.5: Results from the simulation experiment. (top) Percentage overlap over five repair iterations using IterativeRepair (labeled as search) and IterativeBayesRepair (labeled as bayesian) algorithms across three tasks (storytelling, neck exercise, Q&A). (middle) Computation time of the algorithms for the same settings. (bottom) Overlap percentage over five repair iterations for the IterativeBayesRepair algorithm using clean and noisy state corrections for repair. Error bars show standard deviations.

noise of $[-2, 2]$ to every human intention change that occurred when generating the ground truth state (Sec. 7.3.2); total noise duration could at worst equal 37% of task duration.[1] The ± 2 seconds window size was selected based on observing two first time users in an informal study. Repaired parameters were then used to produce a set of FSM output states using the set of FSM input traces in the test dataset. We initialized program parameters so the initial program always produced the missing transition errors. We repeated this procedure 64 times (using the same test dataset), with each run producing a different training input sequence.

Measures

We measured the interaction quality of the repaired programs as the percentage overlap (Eq. 7.3) between the output state trace and the ground truth state traces in the test dataset. Although this overlap had limitations—such as its sensitivity to the interaction length we controlled in this experiment—it captured interaction timing, an important factor in human-robot interaction [84]. We also measured the speed of repair algorithms in milliseconds to gauge the feasibility of repairing a program in front of a real user.

Results

IterativeRepair and IterativeBayesRepair algorithms performed similarly in terms of interaction quality(Fig. 7.5left). Both algorithms' average percentage overlaps increased monotonically over the five iterations, reached average overlaps above 95% by the third iteration, and produced standard deviations lower than 9% by the final iteration in all three tasks. The average computation time for IterativeBayesRepair remained constant over repair iterations, while that for IterativeRepair grew linearly (Fig. 7.5middle). Using the noisy state corrections decreased performance (Fig. 7.5right). Nonetheless, for all three tasks, the final iteration reached 91% mean overlap (with below 14% standard deviation).

[1] Calculated by 2s $\times$ 5 state changes/ 27.02s = 37% (storytelling)

Figure 7.6: Setup for collecting real human data. TabRo is placed on a table and the user sits across the robot.

7.3.3 Human Experiment

To evaluate the full implementation in a realistic setting, we conducted a human experiment with TaRo. Ten participants interacted with a robot for the neck exercise or open-ended Q&A scenarios (Sec. 7.3.1) over four repair iterations.

Robot Platform

We use TaRo for the human experiment. In addition to the basic capabilities of TaRo introduced in Sec. 1.1.3, TaRo uses the camera attached to the monitor and microphone attached to the connected computer to estimate the face poses of a human user using PoseNet [153] and estimate the voice activity level using a custom-built estimator. Both face poses and voice activity inputs are sampled at a synchronized sampling rate of 20hz.

Procedure

The 10 participants (three females) were recruited from the University of Washington (UW) campus community through mailing lists. Their average age was 23.2 (SD = 5.58). Upon

their arrival, the researcher explained the study's purpose, introduced the robot and demonstrated how to interact with it. The first five participants were asked to run the neck exercise and open-ended Q&A programs in order. The last five were asked to do the same in the reverse order. For each task, participants were asked to interact with the robot for four iterations and to reply to the questionnaire after each iteration. Over the four iterations, FSM transition parameters were repaired using IterativeBayesRepair. After each iteration, the desired state traces for IterativeBayesRepair were acquired from participants' recorded button tap inputs using ImplicitStateCorrection (Alg. 1). All transition parameters were initialized to make the robot not respond to users except when they tapped buttons like "Go back."

Measures

Objective measures of program interaction quality included (1) the percentage overlap before and after the repair, and (2) the number of user inputs needed to correct transition errors, i.e., button taps. Percentage overlaps were calculated using the state trace generated with the transition parameters before or after the repair and the desired state trace acquired from the human user in each iteration. We also gathered subjective measures of interaction quality, asking for user agreement with the statements:

- "I had to carry the weight to make the human-robot interaction better." (reverse scale)
- "I trusted the robot to do the right thing at the right time."
- "The robot contributed to the fluency of the interaction."

Available answer choices were 1 (strongly disagree) through 5 (strongly agree). We selected these questions from Hoffman's questionnaire, which was designed to measure the perceived fluency of a human-robot interaction [83]. To investigate the source of potential failures, we asked the open-ended question "What problems did you experience while interacting with

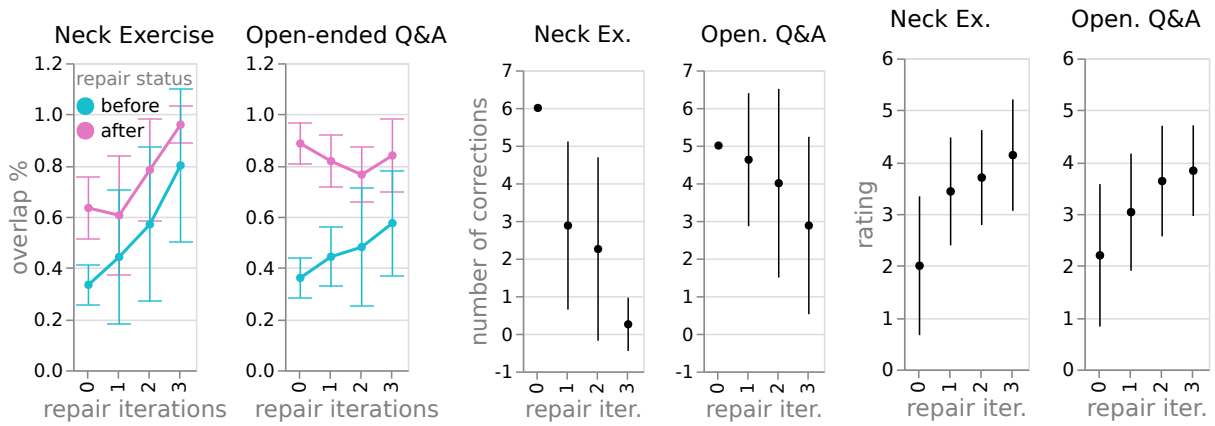


Figure 7.7: Results from the user study of four repair iterations averaged across 10 participants. (left) Percentage overlaps before and after repair. (middle) Average number of interaction corrections (e.g., tapping "Go back") used as implicit feedback for repair. (right) Subjective ratings (reversed as appropriate, with higher values representing more fluency).

the robot during the *taskname* activity?" in addition to the other questions after participants completed each task.

Results

All average objective measures of interaction quality, except after-repair percentage overlaps, increased over the four iterations of the two tasks (Fig. 7.7). We observed (1) a consistent increase in the average before-repair percentage overlap measures, (2) a consistent decrease in the average number of corrective human inputs measures, and (3) a consistent increase in the average subjective ratings between every pair of consecutive iterations. Between the first and last iterations, before-repair percentage overlaps increased from M = 0.33 & SD = 0.08 to M = 0.80 & SD = 0.30 for the neck exercise, and M = 0.36 & SD = 0.08 to M = 0.57 & SD = 0.21 for open-ended Q&As. In both scenarios, initial programs always required participants to tap a button to proceed, i.e., the number of corrective human inputs were 6 and 5, respectively, but the numbers dropped to 0.25 (SD = 0.25) and 2.88 (SD = 2.88), respectively, by the fourth iteration. Subjectively, participants did not initially perceive the

interactions as being fluent (M = 2.00 & SD = 1.34 for neck exercise; M = 2.20 & SD = 1.37 for open-ended Q&A); however, they eventually perceived them as being slightly fluent by the last iteration (M = 4.13 & SD = 1.07 for neck exercise, M = 3.83 & SD = 0.88 for open-ended Q&A).

We analyzed the system logs of the three participants who thought the interaction had not improved by the last iteration. One said of the Q&A task, *"The robot over time got very fast in skipping questions while I'm still talking."* We observed that our approach did not improve interaction quality when unexpected sensor values occurred or the user did not fix incorrect transition errors. For example, some participants frequently moved out of the robot's field of view, while others did not tap the "Go back" button when the robot skipped to the next question; both behaviors caused the algorithm to find ineffective or even counterproductive parameters.

To demonstrate the qualitative differences in human-robot interactions before and after the repair, we present two pairs of interaction traces selected from the user study (Fig. 7.8). In the neck exercise example (above), the program parameters used in the second iteration made the robot incapable of detecting the user following the instruction action. The user noticed this and tapped the "Next" button to make the robot move onto the next state. After the repair in the third iteration, the robot was able to promptly move onto the next state when the user followed the robot's instructions. In the open-ended Q&A example (below), the program parameters used in the second iteration made the robot unable to detect whether the human is not looking at the robot to hold the conversation floor and hence caused the robot to skip to the next question when the human stops speaking. The user then tapped the "Go back" button to continue answering now the previous question. The system used this input to annotate the part of state trace in which the robot accidentally moved onto the next state. After the repair in the fourth iteration, the user again looks away from the robot and stops speaking but the robot waits until the user looks back at the robot to move on to the next question.

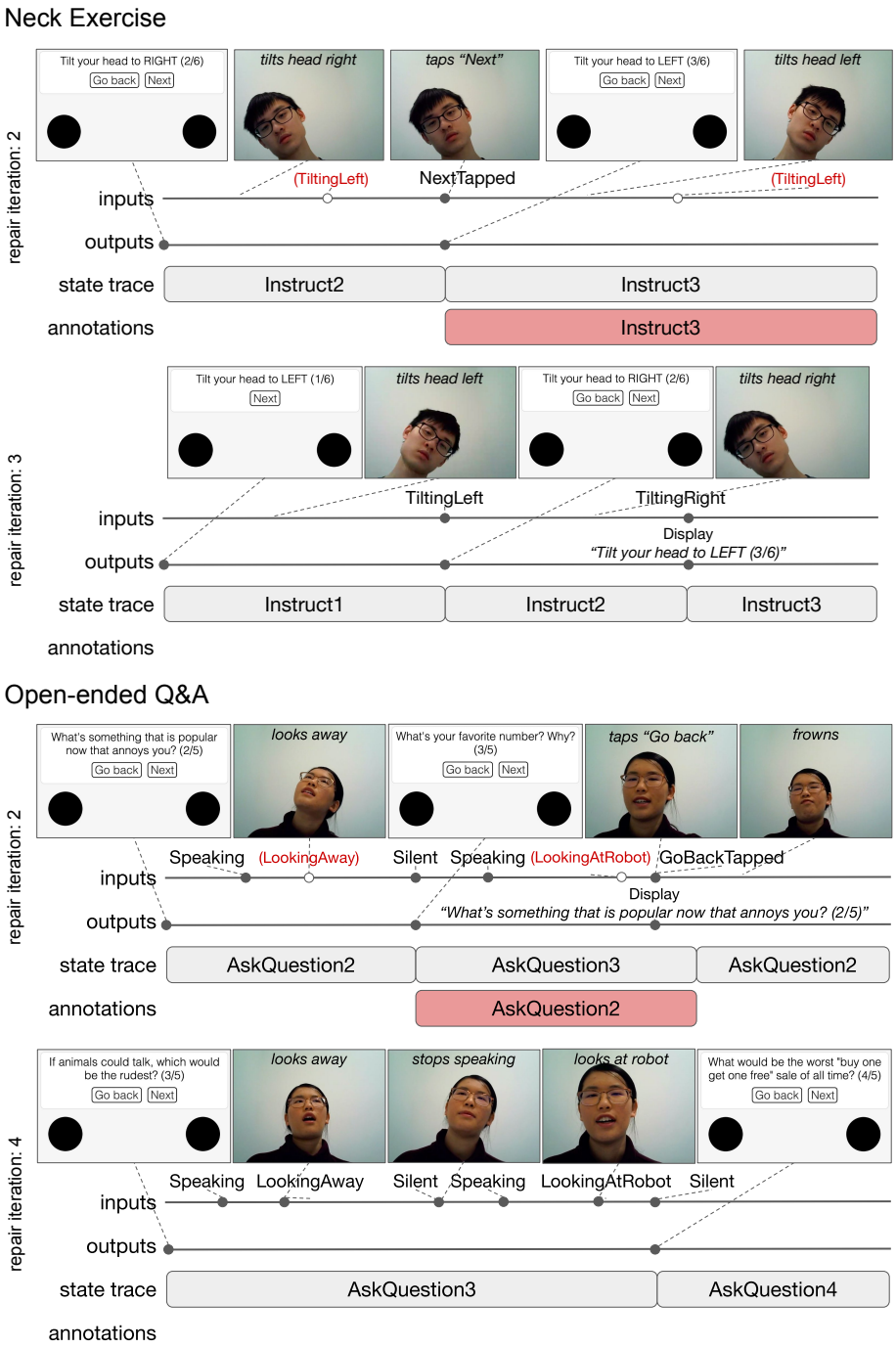


Figure 7.8: Example interaction traces in the neck (top) and Q&A (bottom) scenarios that qualitatively demonstrate the impact of program repair. Undetected human inputs and state annotations are highlighted in red.

Table 7.1: Percentage overlaps before and after the repair, and the number of human inputs measured in the behavior adaptation experiment.

repair iteration	1	2	3	4	5	6
environment description	quiet	quiet	quiet	noisy	noisy	noisy
% overlap before	33.12	99.88	99.92	17.90	57.70	75.42
% overlap after	94.56	99.88	99.92	99.97	99.92	99.93
number of human inputs	5	0	0	5	3	2

Robot Behavior Adaptation Experiment

To demonstrate how our approach handled changes in environment that impact a program's interaction quality, we conducted an experiment involving one user who also participated in the previous human experiment. This user interacted with the robot that ran the open-ended Q&A FSM over six repair iterations; the first three occurred in a quiet room, and the last three in a noisy open indoor area. Between all iterations, we applied IterativeBayesRepair. We measured before and after percentage overlaps and the number of corrective human inputs. Table 7.1 shows these results. The interactive quality of the program quickly improved when the experiment took place in the quiet room (e.g., the required number of human inputs dropped from 5 to 0). The program performed poorly immediately after changing its environment, i.e., the % overlap dropped from 99.82% to 17.90%, and the number of human inputs increased from 0 to 5 but slowly improved in subsequent iterations. Based on analyzing the system log, we found that the improvement over the last three iterations was smaller and took a longer time than that over the first three iterations because (1) the noisy environment produced signals with larger variance, which made the repair more challenging, and (2) the system had to update the prior that was tuned for the first environment, i.e., it had to partially re-learn how to interact with humans.

7.4 Discussion and Future Work

We believe the results demonstrate the usefulness of the proposed approach, with some limitations.

In the human experiment results, we observed that three users did not fix the incorrect transition that occurred during the open-ended Q&A interaction. We anticipate that users will be more likely to provide adversarial feedback as a robot behavior FSM becomes more complex or an input mechanism for corrective feedback becomes more noisy. Hence, it will be necessary to identify robot behavior FSM design patterns or guidelines for creating interactions that avoid problematic feedback for repair tasks and to investigate intuitive recovery methods for resetting inappropriately repaired parameters.

While the proposed algorithms improved interactions measured in our experiments, we expect the following modifications will make them more useful. The proposed algorithm for realizing programmer-free repair, ImplicitStateCorrection, requires the window size parameter to correct the missing transition error. Investigating an effective way to remove this requirement, e.g., by using a credit assignment strategy instead, may help with handling more complex interactions. The complexity of the core IterativeBayesRepair uses exhaustive search, which grows exponentially with respect to the number of transitions, not the number of hole variables. Finding a more scalable inference algorithm or one that does not require any prior distributions for hole variables may increase usability and applicability.

Returning to the goal of our research (Fig. 7.1), we aim to make program repair ultimately benefit end-users who are not proficient software developers. To that end, further research in the workflow for enabling end-users to create and tune robot behaviors by themselves, e.g., by providing a graphical user interface for creating initial FSMs with a list of templates, is imperative. It is also important to further test our approach in the wild and for long-term scenarios where user preferences and environments may change over time to surface and investigate issues that arise.

7.5 Conclusion

This chapter presented an iterative program repair approach for creating robust and fluent social robot programs without painstakingly tuning program parameters. Our approach helps programmers to implement robot programs without complete low-level details and incrementally repair programs over time using corrective feedback provided by the programmer or the robot's user. We examined the feasibility and effectiveness of our approach via two experiments involving human simulators and 10 real human users across three representative social robot use cases; results demonstrate the utility and potential of the proposed approach.

Chapter 8
CONCLUSION

This thesis describes our research on human-centered design approaches and robot systems that let non-experts create interactive applications and behaviors. We demonstrated the creation of two new robot applications for commercial spaces by applying human-centered design approaches (Chapter 3, Chapter 4). Based on our experiences, we described user requirements and design implications for each application. We also presented a series of non-expert-friendly programming systems for creating and tuning interactive robot behaviors. The programming systems enabled exploring of new robot applications (Chapter 5), expressing concurrency (Chapter 6), and improving the interaction quality with minimal programmer effort Chapter 7, with some limitations. I believe that non-expert users will be creating real-world applications for robots in the near future, and I hope my thesis research helps make this significant development possible.

8.0.1 Future Work

Employing Functional Reactive Programming

This research focuses on non-expert robot programming systems that support imperative programming. We began our research with an existing system already being used by novices to create mobile and manipulator robot programs in a short timeframe ([93, 92]); only in retrospect did we realize that the programming paradigm supported by the original programming system adversely affected other important system decisions, such as possible language extensions and choice of user interface. In the future, we plan to investigate other programming paradigms to find one that better supports the requirements for programming interactive robots. Based on our research, we want to explore functional reactive program-

ming (FRP). We anticipate that FRP's ability to express richly interactive, multi-modal interactions via its native data types that model time makes it a strong candidate for interactive robot programming. Researchers already have begun to examine the use of FRP to program robots for educational purposes [154] and the use of visual languages to prototype physical interactions [24].

Exploring Alternative User Interfaces

We did not explore deeply issues concerning the user interface. Based on our research experiences using a block-based visual programming interface with non-roboticist service workers (Chapter 5), we observed that the interface can be tedious and error-prone with long-term use. As user-created programs become more complex, editing them required performing more basic tasks, such as drag-and-drops, zoom-ins (outs); this began to annoy most users. In the future, we plan to explore alternative interfaces for programming interactive robots, such as ones based on virtual reality, augmented reality, or texts. Specifically, we want to investigate a textual interface, which has been suggested by robotics researchers in the past and recently ([127, 203]) but has not been extensively evaluated with novice users in long-term scenarios. Compared to other media-rich programming interfaces, we believe textual interfaces have the potential to connect with a wider group of users of social and service robots because they are accessible and familiar.

Interactive Robot Programming as Program Synthesis

Program synthesis is the task of automatically finding a program that satisfies given user specifications. We believe an interesting research direction for the future is tackling interactive robot behavior authoring problems as program synthesis problems. For example, novice user-created interactive service robot programs need to meet the requirements of multiple stakeholders (Chapter 5). By taking a programming synthesis view, one could represent such requirements as one type of specification, represent the novice user's program as another type of specification, and ask a solver to find a program that meets both specifications.

To the best of our knowledge, a similar idea has been explored for programming smart home devices but not for robots [232]. Program synthesis also expands ways to explore alternative user interfaces (e.g., [167]). User interfaces for program synthesizers are domain-specific languages for creating specifications. Investigating a domain-specific language that captures the common needs of robot programmers is an imperative research topic for the near future. We also anticipate the research potential of technical challenges such as modeling humans as a part of the environment, representing time, and finding algorithms that can handle potentially inconsistent user specifications.

Robot Product Design Techniques in the Wild

Recently, researchers presented design methodologies for developing interactive robot applications in the wild [229, 47, 212]. Compared to the methodologies we employed in Chapter 3 and Chapter 4, the presented methods focus on developing robot products (i.e., applications that provide concrete values) faster (e.g., using entrepreneurial style techniques [177]). Sharing researchers' experiences with challenges and benefits of applying the proposed methods, understanding the requirements and needs of a diverse user cohort (e.g., international users and underrepresented populations), and examining the consequences of using robot systems over the longer-term offer worthwhile future research directions. Finally, we strongly believe that continued research in non-expert, user-friendly programming systems is fundamental in order to make the suggested future research possible.

BIBLIOGRAPHY

[1] Aethon TUG. `http://www.aethon.com/tug/`. [Online; accessed 07-25-2015].

[2] Jonathan Aldrich, David Garlan, Christian Kästner, Claire Le Goues, Anahita Mohseni-Kabir, Ivan Ruchkin, Selva Samuel, Bradley Schmerl, Christopher Steven Timperley, Manuela Veloso, et al. Model-based adaptation for robotics software. *Software*, 36(2):83–90, 2019.

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

[4] Philipp Allgeuer and Sven Behnke. Hierarchical and state-based architectures for robot behavior planning and control. *arXiv preprint arXiv:1809.11067*, 2018.

[5] Sean Andrist, Dan Bohus, Zhou Yu, and Eric Horvitz. Are you messing with me?: querying about the sincerity of interactions in the open world. In *late breaking report, Proc. of HRI*, pages 409–410, 2016.

[6] Sean Andrist, Xiang Zhi Tan, Michael Gleicher, and Bilge Mutlu. Conversational gaze aversion for humanlike robots. In *International Conference on Human-Robot Interaction*, pages 25–32. ACM, 2014.

[7] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 2013.

[8] No Isolation AV1. `https://www.noisolation.com/global/av1/`. [Online; accessed 07-01-2020].

[9] Alper Aydemir, Andrzej Pronobis, Moritz Gobelbecker, and Patric Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 29(4):986–1002, 2013.

[10] Alper Aydemir, K Sjoo, John Folkesson, Andrzej Pronobis, and Patric Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *IEEE International Conference on Robotics and Automation*, pages 2818–2824. IEEE, 2011.

[11] Shiri Azenkot, Catherine Feng, and Maya Cakmak. Enabling building service robots to guide blind people a participatory design approach. In *Proc. of HRI*, pages 3–10, 2016.

[12] Ralf Bachmayer, Susan Humphris, Daniel Fornari, CL Van Dover, Jonathan Howland, Andrew Bowen, Robert Elder, Thomas Crook, Denzel Gleason, William Sellers, et al. Oceanographic research using remotely operated underwater robotic vehicles: Exploration of hydrothermal vent sites on the mid-atlantic ridge at 37 north 32 west. *Marine Technology Society Journal*, 32(3), 1998.

[13] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[14] Emilia I Barakova, Jan CC Gillesen, Bibi EBM Huskens, and Tino Lourens. End-user programming architecture facilitates the uptake of robots in social therapies. *Robotics and Autonomous Systems*, 61(7):704–713, 2013.

[15] Sara L Beckman and Michael Barry. Innovation as a learning process: Embedding design thinking. *California management review*, 50(1):25–56, 2007.

[16] BENBRIA Loop. `www.benbria.com`. [Online; accessed 07-01-2020].

[17] Vincent Berenz and Stefan Schaal. The playful software platform: Reactive programming for orchestrating robotic behavior. *Robotics & Automation Magazine*, 25(3):49–60, 2018.

[18] Hugh Beyer and Karen Holtzblatt. *Contextual design: defining customer-centered systems*. 1997.

[19] Joydeep Biswas and Manuela M Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013.

[20] Jonathan Bohren and Steve Cousins. The smach high-level executive [ros news]. *Robotics & Automation Magazine*, 17(4):18–20, 2010.

[21] Dan Bohus, Sean Andrist, and Mihai Jalobeanu. Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence. In *ACM International Conference on Multimodal Interaction*, pages 493–494. ACM, 2017.

[22] Dan Bohus and Eric Horvitz. Managing human-robot engagement with forecasts and... um... hesitations. In *International Conference on Multimodal Interaction*, pages 2–9. ACM, 2014.

[23] Dan Bohus, Chit W Saw, and Eric Horvitz. Directions robot: in-the-wild experiences and lessons learned. In *Proc. of AAMAS*, pages 637–644, 2014.

[24] bonsai. `https://bonsai-rx.org/`. [Online; accessed 07-01-2020].

[25] Rafael H Bordini, Michael Fisher, and Maarten Sierhuis. Formal verification of human-robot teamwork. In *International conference on Human robot interaction*, pages 267–268. ACM/IEEE, 2009.

[26] Adrian Boteanu, Jacob Arkin, Siddharth Patki, Thomas Howard, and Hadas Kress-Gazit. Robot-initiated specification repair through grounded language interaction. *arXiv preprint arXiv:1710.01417*, 2017.

[27] Cynthia Breazeal. Social robots for health applications. In *Proc. of EMBC*, pages 5368–5371, 2011.

[28] Davide Bresolin, Khaled El-Fakih, Tiziano Villa, and Nina Yevtushenko. Deterministic timed finite state machines: Equivalence checking and expressive power. *arXiv preprint arXiv:1408.5967*, 2014.

[29] Rodney Brooks. A robust layered control system for a mobile robot. *Journal on robotics and automation*, 2(1):14–23, 1986.

[30] Sebastian G Brunner, Franz Steinmetz, Rico Belder, and Andreas Dömel. Rafcon: A graphical tool for engineering complex, robotic tasks. In *International Conference on Intelligent Robots and Systems*, pages 3283–3290. IEEE, 2016.

[31] Nina Buchina, Sherin Kamel, and Emilia Barakova. Design and evaluation of an end-user friendly tool for robot programming. In *International Symposium on Robot and Human Interactive Communication*, pages 185–191. IEEE, 2016.

[32] Hung H Bui, Svetha Venkatesh, and Geoff West. Tracking and surveillance in wide-area spatial environments using the abstract hidden markov model. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):177–196, 2001.

[33] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. Experiences with an interactive museum tour-guide robot. *Artificial intelligence*, 114(1-2):3–55, 1999.

[34] Ryan Calo. The drone as privacy catalyst. *Stanford Law Review Online*, 64, 2011.

[35] Jennifer Casper and Robin R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 33(3):367–385, 2003.

[36] Crystal Chao and Andrea L Thomaz. Timing in multimodal turn-taking interactions: Control and analysis using timed petri nets. *Journal of Human-Robot Interaction*, 1(1):4–25, 2012.

[37] Crystal Chao and Andrea L Thomaz. Controlling social dynamics with a parametrized model of floor regulation. *Journal of Human-Robot Interaction*, 2(1):4–29, 2013.

[38] Michael Chung, Andrzej Pronobis, Maya Cakmak, Dieter Fox, and Rajesh PN Rao. Designing information gathering robots for human-populated environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[39] Michael Jae-Yoon Chung and Maya Cakmak. Exploring the use of robots for gathering customer feedback in the hospitality industry. In *HRI Workshop on Social Robots in the Wild*, 2018.

[40] Michael Jae-Yoon Chung and Maya Cakmak. "how was your stay?": Exploring the use of robots for gathering customer feedback in the hospitality industry. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 947–954. IEEE, 2018.

[41] Michael Jae-Yoon Chung, Justin Huang, Leila Takayama, Tessa Lau, and Maya Cakmak. Iterative design of a system for programming socially interactive service robots. In *Proc. of ICSR*, pages 919–929, 2016.

[42] Michael Jae-Yoon Chung, Andrzej Pronobis, Maya Cakmak, Dieter Fox, and Rajesh PN Rao. Autonomous question answering with mobile robots in human-populated environments. In *Proc. of IROS*, pages 823–830, 2016.

[43] Timothy H Chung, Geoffrey A Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

[44] Anki Cozmo. `https://www.anki.com/en-us/cozmo.html`. [Online; accessed 07-01-2020].

[45] John W Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. 2013.

[46] Brian Demsky and Martin Rinard. Automatic detection and repair of errors in data structures. In *sigplan notices*, volume 38, pages 78–95. ACM, 2003.

[47] Eric C Deng, Bilge Mutlu, and Maja J Matarić. Formalizing the design space and product development cycle for socially interactive robots. In *Workshop on Social Robots in the Wild at the 2018 ACM Conference on Human-Robot Interaction (HRI)*, 2018.

[48] James Diprose, Bruce MacDonald, John Hosking, and Beryl Plimmer. Designing an api at an appropriate abstraction level for programming social robot applications. *Journal of Visual Languages & Computing*, 39:22–40, 2017.

[49] ISO DIS. 9241-210: 2019. ergonomics of human system interaction-part 210: Human-centred design for interactive systems (formerly known as 13407). *International Standardization Organization (ISO). Switzerland*, 2019.

[50] Matthew Dixon, Karen Freeman, and Nicholas Toman. Stop trying to delight your customers. *Harvard Business Review*, 88(7/8):116–122, 2010.

[51] Nicola Doering, Sandra Poeschl, Horst-Michael Gross, Andreas Bley, Christian Martin, and Hans-Joachim Boehme. User-centered design and evaluation of a mobile shopping robot. *International Journal of Social Robotics*, 7(2):203–225, 2015.

[52] Bernard Espiau, Konstantinos Kapellos, and Muriel Jourdan. Formal verification in robotics: Why and how? In *Robotics Research*, pages 225–236. Springer, 1996.

[53] Vanessa Evers, Nuno Menezes, Luis Merino, Dariu Gavrila, Fernando Nabais, Maja Pantic, Paulo Alvito, and Daphne Karreman. The development and real-world deployment of FROG, the fun robotic outdoor guide. In *ACM/IEEE International Conference on Human-robot Interaction (abstract)*, 2014.

[54] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for mobile robots. In *International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.

[55] Juan Fasola and Maja J Matarić. A socially assistive robot exercise coach for the elderly. *Journal of Human-Robot Interaction*, 2(2):3–32, 2013.

[56] Juan Fasola and Maja J Mataric. Interpreting instruction sequences in spatial language discourse with pragmatics towards natural human-robot interaction. In *IEEE International Conference on Robotics and Automation*, 2014.

[57] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[58] Julia Fink, Séverin Lemaignan, Pierre Dillenbourg, Philippe Rétornaz, Florian Vaussard, Alain Berthoud, Francesco Mondada, Florian Wille, and Karmen Franinović. Which robot behavior can motivate children to tidy up their toys?: Design and evaluation of "Ranger". In *Proc. of the International Conference on Human Robot Interaction*, 2014.

[59] Loren Fiore, Duc Fehr, Robot Bodor, Andrew Drenner, Guruprasad Somasundaram, and Nikolaos Papanikolopoulos. Multi-camera human activity monitoring. *Journal of Intelligent and Robotic Systems*, 52(1):5–43, 2008.

[60] John C Flanagan. The critical incident technique. *Psychological bulletin*, 51(4):327, 1954.

[61] Terrence Fong, Illah Nourbakhsh, and Kerstin Dautenhahn. A survey of socially interactive robots. *Robotics and autonomous systems*, 2003.

[62] P-E Forssén, David Meger, Kevin Lai, Scott Helmer, James J Little, and David G Lowe. Informed visual search: Combining attention and object recognition. In *IEEE International Conference on Robotics and Automation*, 2008.

[63] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 1997.

[64] Neil Fraser et al. Blockly: A visual programming editor. *URL: https://code. google. com/p/blockly*, 2013.

[65] Yuxiang Gao and Chien-Ming Huang. Pati: a projection-based augmented tabletop interface for robot programming. In *International Conference on Intelligent User Interfaces*, pages 345–355, 2019.

[66] Jan CC Gillesen, EI Barakova, Bibi EBM Huskens, and Loe MG Feijs. From training to robot behavior: Towards custom scenarios for robotics in training programs for asd. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–7. IEEE, 2011.

[67] D Glas, Satoru Satake, Takayuki Kanda, and Norihiro Hagita. An interaction design framework for social robots. In *Robotics: Science and Systems*, volume 7, page 89, 2012.

[68] Dylan F Glas, Takayuki Kanda, and Hiroshi Ishiguro. Human-robot interaction design using interaction composer eight years of lessons learned. In *International Conference on Human-Robot Interaction*, pages 303–310. ACM/IEEE, 2016.

[69] Rachel Gockley, Allison Bruce, Jodi Forlizzi, Marek Michalowski, Anne Mundell, Stephanie Rosenthal, Brennan Sellner, Reid Simmons, Kevin Snipes, Alan C Schultz, et al. Designing robots for long-term social interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1338–1343. IEEE, 2005.

[70] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[71] Noah D Goodman and Andreas Stuhlmüller. The design and implementation of probabilistic programming languages. `http://dippl.org`, 2014. Accessed: 2019-10-1.

[72] Javi F. Gorostiza and Miguel A. Salichs. End-user programming of a social robot by dialog. *Robotics and Autonomous Systems*, 59(12):1102–1114, 2011.

[73] Stephen Gould, Paul Baumstarck, Morgan Quigley, Andrew Y Ng, and Daphne Koller. Integrating visual and range data for robotic object detection. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications*, 2008.

[74] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 2007.

[75] H-M Gross, H Boehme, Ch Schroeter, Steffen Müller, Alexander Koenig, Erik Einhorn, Ch Martin, Matthias Merten, and Andreas Bley. TOOMAS: interactive shopping guide robots in everyday use-final implementation and experiences from long-term field trials. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[76] Jenna Claire Hammond, Joydeep Biswas, and Arjun Guha. Automatic failure recovery for end-user programs on service mobile robots. *arXiv preprint arXiv:1909.02778*, 2019.

[77] Marc Hanheide, Denise Hebesberger, and Tomas Krajnik. The when, where, and how: an adaptive robotic info-terminal for care home residents–a long-term study. In *Proc. of HRI*, pages 341–349, 2017.

[78] Happy-or-Not. `https://www.happy-or-not.com`. [Online; accessed 07-01-2020].

[79] Mark Harman and Nadia Alshahwan. Automated session data repair for web application regression testing. In *International Conference on Software Testing, Verification, and Validation*, pages 298–307. IEEE, 2008.

[80] Denise Hebesberger, Christian Dondrup, Tobias Koertner, Christoph Gisinger, and Juergen Pripfl. Lessons learned from the deployment of a long-term autonomous robot as companion in physical therapy for older adults with dementia: A mixed methods study. In *Proc. of HRI*, pages 27–34, 2016.

[81] Marcel Heerink, Ben Krose, Vanessa Evers, and Bob Wielinga. Measuring acceptance of an assistive social robot: a suggested toolkit. In *Proc. of RO-MAN*, pages 528–533, 2009.

[82] Sachithra Hemachandra, Thomas Kollar, Nicholas Roy, and Seth Teller. Following and interpreting narrated guided tours. In *IEEE International Conference on Robotics and Automation*, 2011.

[83] Guy Hoffman. Evaluating fluency in human–robot collaboration. *IEEE Transactions on Human-Machine Systems*, 49(3):209–218, 2019.

[84] Guy Hoffman, Maya Cakmak, and Crystal Chao. Timing in human-robot interaction. In *International Conference on Human-robot interaction*, pages 509–510. ACM/IEEE, 2014.

[85] Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based robotic information gathering algorithms. *International Journal of Robotics Research*, page 0278364914533443, 2014.

[86] Jarrett Holtz, Arjun Guha, and Joydeep Biswas. Interactive robot transition repair with smt. In *International Joint Conference on Artificial Intelligence*, pages 4905–4911.

[87] Jarrett Holtz, Arjun Guha, and Joydeep Biswas. Smt-based robot transition repair. *arXiv preprint arXiv:2001.04397*, 2020.

[88] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[89] Chien-Ming Huang, Takamasa Iio, Satoru Satake, and Takayuki Kanda. Modeling and controlling friendliness for an interactive museum robot. In *Robotics: Science and Systems*, 2014.

[90] Chien-Ming Huang and Bilge Mutlu. Robot behavior toolkit: generating effective social behaviors for robots. In *ACM/IEEE international conference on Human-Robot Interaction*, pages 25–32. ACM, 2012.

[91] Justin Huang. *End-to-End Programming Tools for Mobile Manipulator Robots*. PhD thesis, 2018.

[92] Justin Huang and Maya Cakmak. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In *International Conference on Human-Robot Interaction*, pages 453–462. ACM/IEEE, 2017.

[93] Justin Huang, Tessa Lau, and Maya Cakmak. Design and evaluation of a rapid programming system for service robots. In *Proc. of HRI*, pages 295–302, 2016.

[94] Vieira Neto Hugo. *Visual novelty detection for autonomous inspection robots*. PhD thesis, University of Essex, 2006.

[95] Helge Huttenrauch and Kerstin Severinson Eklundh. Fetch-and-carry with cero: observations from a long-term user study with a service robot. In *IEEE International Workshop on Robot and Human Interactive Communication*, 2002.

[96] Adam Jacoff. Search and rescue robotics. In *Springer Handbook of Robotics*. Springer, 2008.

[97] Jibo. `https://jibo.com/`. [Online; accessed 07-01-2020].

[98] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem. Program repair as a game. In *International conference on computer aided verification*, pages 226–238. Springer, 2005.

[99] Dominik Joho and Wolfram Burgard. Searching for objects: Combining multiple cues to object locations using a maximum entropy model. In *IEEE International Conference on Robotics and Automation*, pages 723–728. IEEE, 2010.

[100] MP Joosse, Manja Lohse, and Vanessa Evers. How a guide robot should behave at an airport insights based on observing passengers. *CTIT Technical Report Series*, (TR-CTIT-15-01), 2015.

[101] Takayuki Kanda, Dylan F Glas, Masahiro Shiomi, Hiroshi Ishiguro, and Norihiro Hagita. Who will be the customer?: A social robot that anticipates people's behavior from their trajectories. In *Proc. of UBICOMP*, pages 380–389, 2008.

[102] Takayuki Kanda, Masahiro Shiomi, Zenta Miyashita, Hiroshi Ishiguro, and Norihiro Hagita. An affective guide robot in a shopping mall. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2009.

[103] Takayuki Kanda, Masahiro Shiomi, Zenta Miyashita, Hiroshi Ishiguro, and Norihiro Hagita. A communication robot in a shopping mall. *IEEE Transactions on Robotics*, 26(5):897–913, 2010.

[104] Daphne Karreman, Geke Ludden, and Vanessa Evers. Visiting cultural heritage with a tour guide robot: a user evaluation study in-the-wild. In *Proc. of ICSR*, pages 317–326, 2015.

[105] Daphne E Karreman, Elisabeth MAG van Dijk, and Vanessa Evers. Contextual analysis of human non-verbal guide behaviors to inform the development of FROG, the fun robotic outdoor guide. In *Human Behavior Understanding*. Springer, 2012.

[106] Daphne E Karreman, Elisabeth MAG van Dijk, and Vanessa Evers. Using the visitor experiences for mapping the possibilities of implementing a robotic guide in outdoor sites. In *Proc. of RO-MAN*, pages 1059–1065, 2012.

[107] Gunhee Kim, Woojin Chung, Kyung-Rock Kim, Munsang Kim, Sangmok Han, and Richard H Shinn. The autonomous tour-guide robot jinny. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[108] Kipsu. `www.kipsu.com`. [Online; accessed 07-01-2020].

[109] Kheng Lee Koay, Emrah Akin Sisbot, Dag Sverre Syrdal, Mick L Walters, Kerstin Dautenhahn, and Rachid Alami. Exploratory study of a robot approaching a person in the context of handing over an object. In *AAAI spring symposium: multidisciplinary collaboration for socially assistive robotics*, 2007.

[110] Thomas Kollar and Nicholas Roy. Utilizing object-object and object-scene context when planning to find things. In *IEEE International Conference on Robotics and Automation*, 2009.

[111] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[112] Alap Kshirsagar, Hadas Kress-Gazit, and Guy Hoffman. Specifying and synthesizing human-robot handovers. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2019.

[113] Alyssa Kubota, Emma IC Peterson, Vaishali Rajendren, Hadas Kress-Gazit, and Laurel D Riek. Jessie: Synthesizing social robot behaviors for personalized neurorehabilitation and beyond.

[114] Lars Kunze, Michael Beetz, Manabu Saito, Haseru Azuma, Kei Okada, and Masayuki Inaba. Searching objects in large-scale indoor environments: A decision-theoretic approach. In *IEEE International Conference on Robotics and Automation*, pages 4385–4390. IEEE, 2012.

[115] Min Kyung Lee, Jodi Forlizzi, Paul E Rybski, Frederick Crabbe, Wayne Chung, Josh Finkle, Eric Glaser, and Sara Kiesler. The snackbot: documenting the design of a robot for long-term human-robot interaction. In *Proc. of HRI*, pages 7–14, 2009.

[116] Min Kyung Lee, Sara Kiesler, and Jodi Forlizzi. Receptionist or information kiosk: how do people talk with a robot? In *ACM Conference on Computer Supported Cooperative Work*, 2010.

[117] Min Kyung Lee, Sara Kiesler, Jodi Forlizzi, and Paul Rybski. Ripple effects of an embedded social agent: a field study of a social robot in the workplace. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, 2012.

[118] Min Kyung Lee and Maxim Makatchev. How do people talk with a robot?: an analysis of human-robot dialogues in the real world. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, pages 3769–3774. ACM, 2009.

[119] Nicola Leonardi, Marco Manca, Fabio Paternò, and Carmen Santoro. Trigger-action programming for personalising humanoid robot behaviour. In *Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[120] Terrence J Levesque and Gordon HG McDougall. Service problems and recovery strategies: an experiment. *Canadian Journal of Administrative Sciences*, 17(1):20–37, 2000.

[121] Wenchao Li, Dorsa Sadigh, S Shankar Sastry, and Sanjit A Seshia. Synthesis for human-in-the-loop control systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 470–484, 2014.

[122] Timm Linder, S. Breuers, and Kai O Arras. On multi-modal people tracking from mobile platforms in very crowded and dynamic environments. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, 2016.

[123] Junbin Liu, Clinton Fookes, Tim Wark, and Sridha Sridharan. On the statistical determination of optimal camera configurations in large scale surveillance networks. In *Computer Vision–ECCV 2012*, pages 44–57. Springer, 2012.

[124] Manja Lohse, Frederic Siepmann, and Sven Wachsmuth. A modeling framework for user-driven iterative design of autonomous systems. *International journal of social robotics*, 6(1):121–139, 2014.

[125] Fan Long and Martin Rinard. Staged program repair with condition synthesis. In *Joint Meeting on Foundations of Software Engineering*, pages 166–178, 2015.

[126] Malte Lorbach, Sebastian Hofer, and Oliver Brock. Prior-assisted propagation of spatial information for object search. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2904–2909. IEEE, 2014.

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

[128] David V Lu and William D Smart. Towards more efficient navigation for robots and humans. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1707–1713. IEEE, 2013.

[129] Catalia Health. `http://www.cataliahealth.com/`. [Online; accessed 07-01-2020].

[130] Vijay Mahadevan, Weixin Li, Viral Bhalodia, and Nuno Vasconcelos. Anomaly detection in crowded scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1975–1981. IEEE, 2010.

[131] Sonal Mahajan, Abdulmajeed Alameer, Phil McMinn, and William GJ Halfond. Xfix: an automated tool for the repair of layout cross browser issues. In *International Symposium on Software Testing and Analysis*, pages 368–371. ACM, 2017.

[132] Marco Manca, Fabio Paternò, and Carmen Santoro. Analyzing trigger-action programming for personalization of robot behaviour in iot environments. In *International Symposium on End User Development*, pages 100–114, 2019.

[133] Spyros Maniatopoulos, Philipp Schillinger, Vitchyr Pong, David C Conner, and Hadas Kress-Gazit. Reactive high-level behavior synthesis for an atlas humanoid robot. In *International Conference on Robotics and Automation*, pages 4192–4199. IEEE, 2016.

[134] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014.

[135] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon. 2010.

[136] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a unified behavior trees framework for robot control. In *International Conference on Robotics and Automation*, pages 5420–5427. IEEE, 2014.

[137] Carlos Mateo, Alberto Brunete, Ernesto Gambao, and Miguel Hernando. Hammer: An android based application for end-user industrial robot programming. In *International Conference on Mechatronic and Embedded SystemsApplications*, pages 1–6. IEEE/ASME, 2014.

[138] Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 251–258. IEEE Press, 2010.

[139] Elena R Messina and Adam S Jacoff. Measuring the performance of urban search and rescue robots. In *IEEE Conference on Technologies for Homeland Security 2007*, pages 28–33, 2007.

[140] MetraLabs Scitos G5. `http://metralabs.com/index.php?option=com_content&view=article&id=70&Itemid=64`. [Online; accessed 07-25-2015].

[141] Scott A Miller, Zachary A Harris, and Edwin KP Chong. A POMDP framework for coordinated guidance of autonomous uavs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009:2, 2009.

[142] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, and Jon Timmis. Automatic property checking of robotic applications. In *International Conference on Intelligent Robots and Systems*, pages 3869–3876. IEEE/RSJ, 2017.

[143] Kazuyuki Morioka, Joo-Ho Lee, and Hideki Hashimoto. Human-following mobile robot in a distributed intelligent sensor network. *IEEE Transactions on Industrial Electronics*, 51(1):229–237, 2004.

[144] Lilia Moshkina, Susan Trickett, and J Gregory Trafton. Social engagement in public places: a tale of one robot. In *Proc. of HRI*, pages 382–389, 2014.

[145] Caio Mucchiani, Suneet Sharma, Megan Johnson, Justine Sefcik, Nicholas Vivio, Justin Huang, Pamela Cacchione, Michelle J Johnson, Roshan Rai, Adrian Canoso, Tessa Lau, and Mark Yim. Evaluating older adults interaction with a mobile assistive robot. In *Proc. of IROS*, 2017.

[146] Bilge Mutlu and Jodi Forlizzi. Robots in organizations: the role of workflow, social, and environmental factors in human-robot interaction. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2008.

[147] Srinivas Nedunuri, Sailesh Prabhu, Mark Moll, Swarat Chaudhuri, and LE Kavraki. Smt-based synthesis of integrated task and motion plans for mobile manipulation. In *International Conference on Robotics and Automation*. IEEE, 2014.

[148] Nest Nest Cam. `https://nest.com/camera/meet-nest-cam/`. [Online; accessed 07-25-2015].

[149] Hai Nguyen, Matei Ciocarlie, Kaijen Hsiao, and Charles C Kemp. Ros commander (rosco): Behavior creation for home robots. In *International Conference on Robotics and Automation*, pages 467–474. IEEE, 2013.

[150] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. Semfix: Program repair via semantic analysis. In *International Conference on Software Engineering*, pages 772–781. IEEE, 2013.

[151] Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

[152] Akihisa Ohya, Yousuke Nagumo, and Youhei Gibo. Intelligent escort robot moving together with human-methods for human position recognition. In *International Conference on Soft Computing and Intelligent Systems*, 2002.

[153] D Oved. Real-time human pose estimation in the browser with tensorflow.js. *TensorFlow Medium, May*, 2018.

[154] Hugo Pacheco and Nuno Macedo. Rosy: An elegant language to teach the pure reactive nature of robot programming. *arXiv preprint arXiv:1911.03262*, 2019.

[155] Caroline Pantofaru and Leila Takayama. Need finding: A tool for directing robotics research and development. In *RSS 2011 Workshop on perspectives and contributions to robotics from the human sciences*, 2011.

[156] Caroline Pantofaru, Leila Takayama, Tully Foote, and Bianca Soto. Exploring the role of robots in home organization. In *Proc. of HRI*, pages 327–334, 2012.

[157] Vivek Paramasivam, Justin Huang, Sarah Elliott, and Maya Cakmak. Computer science outreach with end-user robot-programming tools. In *Technical Symposium on Computer Science Education*, pages 447–452. ACM, 2017.

[158] Hae Won Park, Mirko Gelsomini, Jin Joo Lee, and Cynthia Breazeal. Telling stories to robots: The effect of backchanneling on a child's storytelling. In *International Conference on Human-Robot Interaction*, pages 100–108. IEEE, 2017.

[159] Hae Won Park, Mirko Gelsomini, Jin Joo Lee, Tonghui Zhu, and Cynthia Breazeal. Backchannel opportunity prediction for social robot listeners. In *International Conference on Robotics and Automation*, pages 2308–2314. IEEE, 2017.

[160] Jongkyeong Park and Gerard Jounghyun Kim. Robots with projectors: an alternative to anthropomorphic hri. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2009.

[161] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D Hager. Costar: Instructing collaborative robots with behavior trees and vision. In *Onternational Conference on Robotics and Automation*, pages 564–571. IEEE, 2017.

[162] Chris Paxton, Felix Jonathan, Andrew Hundt, Bilge Mutlu, and Gregory D Hager. Evaluating methods for end-user creation of robot task plans. In *International Conference on Intelligent Robots and Systems*, pages 6086–6092. IEEE, 2018.

[163] Vittorio Perera and Manuela Veloso. Handling complex commands as service robot task requests. 2015.

[164] JM Perez, Fernando Caballero, and Luis Merino. Integration of Monte Carlo Localization and place recognition for reliable long-term robot localization. In *IEEE International Conference on Autonomous Robot Systems and Competitions*, 2014.

[165] Roland Philippsen and Roland Siegwart. Smooth and efficient obstacle avoidance for a tour guide robot. In *IEEE International Conference on Robotics and Automation*, 2003.

[166] Roberto Pinillos, Samuel Marcos, Raul Feliz, Eduardo Zalama, and Jaime Gómez-García-Bermejo. Long-term assessment of a service robot in a hotel environment. *Robotics and Autonomous Systems*, 79:40–57, 2016.

[167] David Porfirio, Evan Fisher, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. Bodystorming human-robot interactions. In *Symposium on User Interface Software and Technology*, 2019.

[168] David Porfirio, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. Authoring and verifying human-robot interactions. In *Symposium on User Interface Software and Technology*, pages 75–86. ACM, 2018.

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

[170] Mary C Potter, Brad Wyble, Carl Erick Hagmann, and Emily S McCourt. Detecting meaning in rsvp at 13 ms per picture. *Attention, Perception, & Psychophysics*, 2014.

[171] Andrzej Pronobis and Patric Jensfelt. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *IEEE International Conference on Robotics and Automation*, pages 3515–3522. IEEE, 2012.

[172] Mattia Racca, Ville Kyrki, and Maya Cakmak. Interactive tuning of robot program parameters via expected divergence maximization.

[173] Savioke. `https://www.savioke.com/`. [Online; accessed 07-01-2020].

[174] Tripadvisor. `www.tripadvisor.com`. [Online; accessed 07-01-2020].

[175] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[176] Charles Rich, Brett Ponsler, Aaron Holroyd, and Candace L Sidner. Recognizing engagement in human-robot interaction. In *International Conference on Human-Robot Interaction*, pages 375–382. ACM/IEEE, 2010.

[177] Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Currency, 2011.

[178] Stephanie Rosenthal, Joydeep Biswas, and Manuela Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 915–922. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[179] Astrid M Rosenthal-von der Pütten, Nicole C Krämer, Laura Hoffmann, Sabrina Sobieraj, and Sabrina C Eimler. An experimental study on emotional reactions towards a robot. *International Journal of Social Robotics*, 5(1):17–34, 2013.

[180] Paul E Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M Veloso. Interactive robot task training through dialog and demonstration. In *International Conference on Human-Robot Interaction*, pages 49–56. ACM/IEEE, 2007.

[181] Selma Sabanovic, Marek P Michalowski, and Reid Simmons. Robots in the wild: Observing human-robot social interaction outside the lab. In *IEEE International Workshop on Advanced Motion Control 2006*, pages 596–601, 2006.

[182] Mehdi Samadi, Thomas Kollar, and Manuela M Veloso. Using the web to interactively learn to find objects. In *AAAI Conference on Artificial Intelligence*, 2012.

[183] Satoru Satake, Takayuki Kanda, Dylan F Glas, Michita Imai, Hiroshi Ishiguro, and Norihiro Hagita. How to approach humans?-strategies for social robots to initiate interaction. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2009.

[184] Allison Sauppé and Bilge Mutlu. Robot deictics: How gesture and context shape referential communication. In *International Conference on Human-Robot Interaction*, pages 342–349. ACM, 2014.

[185] Savioke SaviOne. `http://www.savioke.com`. [Online; accessed 07-25-2015].

[186] Brian Scassellati. How social robots will help us to diagnose, treat, and understand autism. In *Robotics research*, pages 552–563. 2007.

[187] Philipp Schillinger, Stefan Kohlbrecher, and Oskar von Stryk. Human-robot collaborative high-level control with application to rescue robotics. In *International Conference on Robotics and Automation*, pages 2796–2802. IEEE, 2016.

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

[189] Nir Shavit. Data structures in the multicore age. *Communications of the ACM*, 54(3):76–84, 2011.

[190] Chao Shi, Masahiro Shiomi, Christian Smith, Takayuki Kanda, and Hiroshi Ishiguro. A model of distributional handing interaction for a mobile robot. In *Proc. of RSS*, pages 24–28, 2013.

[191] Masahiro Shiomi, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. A larger audience, please!: encouraging people to listen to a guide robot. In *Proc. of HRI*, pages 31–38, 2010.

[192] Hedvig Sidenbladh, Danica Kragic, and Henrik I Christensen. A person following behaviour for a mobile robot. In *IEEE International Conference on Robotics and Automation*, 1999.

[193] Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, pages 707–755, 2009.

[194] Armando Solar-Lezama. Program sketching. *International Journal on Software Tools for Technology Transfer*, 15(5-6):475–495, 2013.

[195] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM Sigplan Notices*, 41(11):404–415, 2006.

[196] Xuan Song, Xiaowei Shao, Quanshi Zhang, Ryosuke Shibasaki, Huijing Zhao, and Hongbin Zha. Laser-based intelligent surveillance and abnormality detection in extremely crowded scenarios. In *IEEE International Conference on Robotics and Automation*, pages 2170–2176. IEEE, 2012.

[197] André Staltz. Cycle.js. `https://cycle.js.org/`, 2016. Accessed: 2019-10-1.

[198] Franz Steinmetz, Annika Wollschläger, and Roman Weitschat. Razer-a human-robot interface for visual task-level programming and intuitive skill parameterization. *Robotics and Automation Letters*, 3(3):1362–1369, 2018.

[199] Maj Stenmark, Mathias Haage, and Elin Anna Topp. Simplified programming of reusable skills on a safe industrial robot: Prototype and evaluation. In *International Conference on Human-Robot Interaction*, pages 463–472. ACM/IEEE, 2017.

[200] Jennifer C Stern, Brad Sutter, Caroline Freissinet, Rafael Navarro-González, Christopher P McKay, P Douglas Archer, Arnaud Buch, Anna E Brunner, Patrice Coll, Jennifer L Eigenbrode, et al. Evidence for indigenous nitrogen in sedimentary and aeolian deposits from the curiosity rover investigations at gale crater, mars. *Proc. of the National Academy of Science*, 112(14):4245–4250, 2015.

[201] BeamPro. `https://www.suitabletech.com/beampro/`. [Online; accessed 07-25-2015].

[202] JaYoung Sung, Henrik I Christensen, and Rebecca Grinter. Robots in the wild: understanding long-term use. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2009.

[203] Craig J Sutherland and Bruce MacDonald. Robolang: A simple domain specific language to script robot interactions. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 265–270. IEEE, 2019.

[204] Takahiro Suzuki, Fumihiro Bessho, Tatsuya Harada, and Yasuo Kuniyoshi. Visual anomaly detection under temporal and spatial non-uniformity for news finding robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1214–1220. IEEE, 2011.

[205] Stefanie Tellex, Ross Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. In *Robotics Science and Systems Conference*, 2014.

[206] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Approaching the symbol grounding problem with probabilistic graphical models. *AI Magazine*.

[207] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI Conference on Artificial Intelligence*, 2011.

[208] Moritz Tenorth and Michael Beetz. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *International Journal of Robotics Research*, 32(5):566–590, 2013.

[209] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, et al. MINERVA: A second-generation museum tour-guide robot. In *IEEE International Conference on Robotics and Automation*, 1999.

[210] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[211] Sebastian Thrun, Jamie Schulte, and Chuck Rosenberg. Interaction with mobile robots in public places. *IEEE Intelligent Systems*, pages 7–11, 2000.

[212] Meg Tonkin, Jonathan Vitale, Sarita Herse, Mary-Anne Williams, William Judge, and Xun Wang. Design methodology for the ux of hri: A field study of a commercial social robot at an airport. In *Proc. of HRI*, pages 407–415. ACM, 2018.

[213] Peter Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. Robot navigation in dense human crowds: the case for cooperation. In *IEEE International Conference on Robotics and Automation*, 2013.

[214] Rudolph Triebel, KO Arras, Rachid Alami, Lucas Beyer, Stefan Breuers, Raja Chatila, Mohamed Chetouani, Daniel Cremers, Vanessa Evers, Michelangelo Fiore, et al. SPENCER: a socially aware service robot for passenger guidance and help in busy airports. 2015.

[215] Walter F Truszkowski, Michael G Hinchey, James L Rash, and Christopher A Rouff. Autonomous and autonomic systems: A paradigm for future space exploration missions. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(3):279–291, 2006.

[216] Jana Tumova and Dimos V Dimarogonas. Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, 70:239–248, 2016.

[217] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *International Journal of Robotics Research*, 2012.

[218] Niek R van Ulzen, Claudine JC Lamoth, Andreas Daffertshofer, Gün R Semin, and Peter J Beek. Characteristics of instructed and uninstructed interpersonal coordination while walking side-by-side. *Neuroscience letters*, 432(2):88–93, 2008.

[219] Vecna QC Bot. `http://www.vecna.com/product/qc-bot-base-model/`. [Online; accessed 07-25-2015].

[220] Javier Velez, Garrett Hemann, Albert S Huang, Ingmar Posner, and Nicholas Roy. Planning to perceive: Exploiting mobility for robust object detection. In *International Conference on Automated Planning and Scheduling*, 2011.

[221] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. CoBots: robust symbiotic autonomous mobile service robots. In *International Conference on Artificial Intelligence*, 2015.

[222] Manuela Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Thomas Kollar, Cetin Mericli, Mehdi Samadi, Susana Brandao, and Rodrigo Ventura. Cobots: Collaborative robots servicing multi-floor buildings. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[223] Antonio W Vieira, Paulo LJ Drews, and Mario FM Campos. Spatial density patterns for efficient change detection in 3d environment for autonomous surveillance robots. *IEEE Transactions on Automation Science and Engineering*, 11(3):766–774, 2014.

[224] Rebecca A Walpole and Margaret M Burnett. Supporting reuse of evolving visual code. In *Proceedings. 1997 IEEE Symposium on Visual Languages (Cat. No. 97TB100180)*, pages 68–75. IEEE, 1997.

[225] Matt Webster, Clare Dixon, Michael Fisher, Maha Salem, Joe Saunders, Kheng Lee Koay, and Kerstin Dautenhahn. Formal verification of an autonomous personal robotic assistant. In *AAAI Spring Symposium Series*, 2014.

[226] Matt Webster, Clare Dixon, Michael Fisher, Maha Salem, Joe Saunders, Kheng Lee Koay, Kerstin Dautenhahn, and Joan Saez-Pons. Toward reliable autonomous robotic assistants through formal verification: a case study. *Transactions on Human-Machine Systems*, 46(2):186–196, 2015.

[227] Julie Weed. Checking in after checkout. *The New York Times*, May 2013.

[228] Why Indoor Robots for Commercial Spaces Are the Next Big Thing in Robotics. `https://spectrum.ieee.org/automaton/robotics/robotics-hardware/indoor-robots-for-commercial-spaces`. [Online; accessed 07-01-2020].

[229] Matthew Willis. Human centered robotics: designing valuable experiences for social robots. In *Proceedings of HRI2018 Workshop (Social Robots in the Wild). ACM, New York*, 2018.

[230] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. Eliminating timing side-channel leaks using program repair. In *International Symposium on Software Testing and Analysis*, pages 15–26. ACM, 2018.

[231] Atef Ben Youssef, Chloe Clavel, and Slim Essid. Early detection of user engagement breakdown in spontaneous human-humanoid interaction. *Transactions on Affective Computing*, 2019.

[232] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. Autotap: synthesizing and repairing trigger-action programs using ltl properties. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 281–291. IEEE, 2019.

[233] Shiqi Zhang and Peter Stone. Corpp: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *AAAI Conference on Artificial Intelligence*, 2015.