Contents lists available at ScienceDirect

International Journal of Human - Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

FritzBot: A data-driven conversational agent for physical-computing system design

Taizhou Chen, Lantian Xu, Kening Zhu [*]

School of Creative Media, City University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
Natural-language interaction
User interface
Physical computing
Design
BiLSTM
CRF

ABSTRACT

Creating physical-computing systems, especially selecting correct electronic components, assembling the circuit, and implementing the program, can be challenging for novice users. In this paper, we present FritzBot, a data-driven conversational agent supporting novice users on creating physical-computing systems through natural-language interaction. FritzBot is built upon the structure of a BiLSTM-CRF (bi-directional Long Short-term Memory Network and Conditional Random Field) neural network, as a plug-in for Fritzing. The neural network is trained on a lexical circuit-event database derived from 152 students' reports on their physical-computing course projects. By processing the user's textual description on his/her physical-computing idea, FritzBot can extract the causal relationships between the input and the output events, identify the corresponding electronic components, and generate the Arduino-based circuit and the code along with the step-by-step construction guidelines. Our user study shows that compared to the original Arduino software and the circuit-autocompletion software available in the commercial market, FritzBot significantly shortens the time spent, reduces the perceived workload, and enhances the satisfaction/joy for inexperienced users on designing and prototyping physical-computing systems.

1. Introduction

Creating physical-computing systems through circuit design and micro-controller programming has became a common practice permeating into various fields such as maker activities, STEM education, new media art, product design, and so on. There have been various hardware platforms (Arduino, 2019; Bea, 2019; Raspberry Pi, 2019) and software (Knörig et al., 2009; Tinkercad, 2019) supporting novice users with minimal technical knowledge to create functional electronic circuits. While these tools have lowered the entry bar of physical computing, they still require technical background to a certain extend, such as the knowledge of electrical theory, a large library of components, and how to use them. Along with the rapid-prototyping electronic platforms, the specially packaged programming languages/libraries and the visual programming languages (e.g. Scratch, 2019) can simplify the coding processes. However, they still require a basic level of programming knowledge (e.g. syntax, storage, control flow, Boolean logic, etc.), which many designers, especially students, may not have. Recent research has shown that it was challenging for most users to construct a circuit with correct code for a physical-computing task, even for those who have a technical background (Mellisy et al., 2016; Waterhouse, 2016).

While existing tools for circuit auto-completion (Circuito.io, 2019; Lo et al., 2019) can simplify the process of circuit design, these systems still require users to decide/choose what components to use at the very beginning. Though novice users often have high-level concepts/ideas about what they want to make, it could be still challenging for them, without prior technical knowledge/experience, to decide what electronic components to use (Mellisy et al., 2016). In addition, these systems do not provide assistance on micro-controller programming to make the circuit actually functional as desired. There is also research focusing on circuit and code generation through dragging and dropping logical action blocks (Anderson et al., 2017). However, it still requires users to have prerequisite knowledge of programming logic. Furthermore, the aforementioned systems are all GUI-based, while research shows that combining both graphical widget interaction and natural-language interaction can enhance the system capability than either technology could individually (Cohen, 1992). More recently, researchers investigated leveraging the voice user interface to support circuit prototyping (Kim et al., 2019). Yet, it still requires users to have basic knowledge of circuit principle and programming logic to describe

* Corresponding author.
E-mail address: keninzhu@cityu.edu.hk (K. Zhu).

https://doi.org/10.1016/j.ijhcs.2021.102699
Received 14 August 2020; Received in revised form 15 July 2021; Accepted 27 July 2021
Available online 6 August 2021

their ideas with a pre-defined technical format.

In this paper, we present FritzBot, a BiLSTM-CRF-based conversational agent offering assistance for novice users on circuit construction and programming through natural-language interaction. After analyzing 152 student reports from an undergraduate physical-computing course in a local art school, we find that physical-computer concepts are mostly described in the form of cause/input and effect/output, such as *"When the user <e1>swings </e1>the baton, a bird will <e2>sing </e2>a song."* (*e1* and *e2* indicate the input and the output events, respectively), to indicate the input and the output elements. With this consideration, the conversational engine of FritzBot is developed based on a BiLSTM-CRF neural network to identify the input events and the output events in users' textual descriptions. When a user enters the textual description of his/her physical-computing idea, FritzBot will extract the causal relationship, identify the input and the output components, and generate the corresponding Arduino-based circuit and code along with the construction guidelines. Meanwhile, FritzBot will generate textual responses to guide the user to improve his/her concept if missing elements are found. Our between-subjects user study with twenty-four art/design students shows that compared to the original Arduino software and the circuit-autocompletion software in the market, FritzBot could significantly reduce the time spent and the perceived workload, and enhance the user satification/joy for the physical-computing task.

The primary contributions of this paper are:

1) We construct a lexical circuit-event database for modeling novice users' natural-language behaviors on physical-computing system design and development.
2) For proof of concept, we develop FrtitzBot, a BiLSTM-CRF-based system leveraging natural-language interaction for designing and creating Arduino-based physical-computing systems. The data set and the source code for model training are available at https://github.com/taizhouchen/FritzBot.
3) We conduct a user study showing the effectiveness of FritzBot on supporting novice users' physical-computing task through natural-language interaction.

2. Related work

In this session, we will discuss prior works on the conversational user interfaces for creative tasks and the interactive supporting tools for circuit design and prototyping. We will also discuss existing works on the deep-learning model for the named-entity recognition task in natural-language processing, which is the core algorithm for FritzBot.

2.1. Conversational user interfaces

Natural-language-based conversational user interface have been widely researched in the past few decades in several domains, including smartphone assistant (Bixby, 2019; Li et al., 2017; 2018; 2019), TV assistant (Coelho et al., 2011), public art creation (Fernando et al., 2009), image editing (Laput et al., 2013), web automation (Lau et al., 2010), vehicle interface (Lin et al., 2018; Sung et al., 2019), and database management (Thanisch, 1995). Before the era of machine learning for natural-language processing, controlled natural language (CNL) (Kuhn, 2014) was widely applied in various domains. The recognition of CNL was achieved by constructing subsets of natural languages that are obtained by restricting the grammar and vocabulary in order to reduce or eliminate ambiguity and complexity. However, the specific keywords and grammars may increase the users' learning efforts (Williams et al., 2014). FritzBot was initially inspired by GameChangineer (Zhan and Hsiao, 2019), a platform that allows users to create computer games through natural-language-based programming. It extracts the game mechanism from an English sentence with per-defined keywords and sentence formats. Similarly, Vajra (Schlegel et al., 2019) proposed an end-user programming paradigm for Python. It allows users to compose Python programs through natural-language description. PixelTone (Laput et al., 2013) combines speech and direct manipulation for image editing. The user study shows that the integration of natural language in the task of image editing can effectively improve the working efficiency. Recently, Srinivasan et al. (2019) introduced an image-editing method combining both touch and speech inputs. While the aforementioned researches showed the benefits of natural-language interaction on various creative processes, the potential of leveraging natural-language interaction on supporting physical-computing prototyping tasks is not fully explored yet. To construct a physical-computing system, like in other creative tasks, novice users often can verbally describe their ideas, but may not know which components to use. Built upon the framework of BiLSTM-CRF neural network, FritzBot supports novice users, such as inexperienced designers and artists, to create a functional physical-computing system by simply describing their ideas in natural language.

2.2. Circuit-design and -prototyping interface

Many software have been developed to facilitate breadboard circuit design. Fritzing (Knörig et al., 2009) offers a drag-and-drop interface for both novice and experienced users to create virtual breadboard circuits using common electronic components. As an open-source software, Fritzing has been widely used in makers' community. Virtual Breadboard (VirtualBreadboard, 2019) and TinkerCAD (Tinkercad, 2019) provide similar functionalities as Fritzing does, but with a build-in simulator allowing user to debug their circuit and code. Circuito.io (2019) could automatically completes the circuit connection after user drag and drop the components. Similarly, AutoFritz (Lo et al., 2019) provide a circuit auto-completion function by analysing online Fritzing projects. Although these tools could give users guidelines for connecting components, it is common that some novice users have no idea of what component they should use to achieve the desired functions. Moreover, composing computer programs for micro-controller is another challenge for novice users who want to create physical-computing systems. A recent study shows that only six out of twenty participants could successfully complete both hardware and software parts of a simple physical computing task (Waterhouse, 2016). Trigger-Action-Circuits (Anderson et al., 2017) simplifies the circuit logic into a high-level behavioral description, which is easier to understand. However, it still requires users to have basic knowledge of computer logic. Wu and Yang (2019) develop Proxino, a tool that blends virtual and physical worlds for prototyping circuits. While these interfaces provide different levels of support for prototyping physical-computing systems, users sometimes still have to obtain basic, sometimes even advanced, circuit and programming knowledge. More recently, Kim et al. developed HeyTeddy (Kim et al., 2019), a voice interface for functional circuit prototyping. Their study proved that with the support of voice input, the circuit prototyping efficiency could be increased. Although HeyTeddy provides a similar idea as FritzBot, that is, introducing natural language interaction for circuit prototyping, HeyTeddy still requires users to have basic knowledge of circuit and logical programming to describe the desired features in a pre-defined technical format, while FritzBot was designed for more novice users. Taking a hand-waving robot as an example, HeyTeddy user needs to identify the key component (i.e. a servo motor here), wire the component circuit, and provide the voice command such as "write high/low to pin#"; In FritzBot, the user just need to enter a natural sentence like "I want to make a robot that waves its hand.", and FritzBot dynamically generates the corresponding Arduino-based circuit and code with the support of the lexical circuit-event database and the machine-learning model. In addition, voice-based user interfaces may not be conducive to privacy and noisy environments, such as a classroom with many students.

2.3. Named-entity recognition for natural-language processing

The conversational engine of FritzBot was built upon the algorithm of named-entity recognition for natural-language processing. In this work, we form our database by extracting all the cause-effect sentences from the students' physical-computing project reports. The essential task of Fritzbot's conversational engine is to recognize the cause phrase entities as the input events and the effect phrase entities as the output events, from the user-input sentence which can be modeled as a sequence of words. Most existing sequence-tagging models are based on the linear statistical models, mainly including Hidden Markov Models (HMMs) (Rabiner and Juang, 1986) and conditional random fields (CRFs) (Lafferty et al., 2001). With the emerging technology of deep machine learning, recurrent neural networks (RNNs) were designed to operate on sequential data, but they tend to be biased towards their most recent inputs in the sequence (e.g., neighboring words in a sentence) (Yoshua Bengio et al., 1994). This makes this type of model not so suitable for natural-language processing. Long Short-term Memory Network (LSTM) (Hochreiter and Schmidhuber, 1997) and bidirectional LSTM (BiLSTM) (Schuster and Paliwal, 1997) were proposed to capture long-range dependencies in the data sequence, and it has been proved to outperform traditional RNNs architecture in video processing (Yue-Hei Ng et al., 2015), sketch encoding (Ha and Eck, 2017), and sensor signal processing (Huang et al., 2018). Named-entity recognition task, namely sequence tagging task, was highly relied on both past features (previous few words) and future features (next few words) in the input sequence. Combining BiLSTM and CRF proved to perform very well on named-entity recognition task (Huang et al., 2015; Lample et al., 2016). In FritzBot, we adopt the technique of BiLSTM + CRF (Huang et al., 2015), to extract the causal relationships in the user's natural-language description of his/her physical-computing idea.

3. Natural language and circuit prototyping

To understand how natural-language interaction can be used to support the task of creating physical-computing systems, we first study how novice users, mainly art/design undergraduate students, describe their physical-computing ideas in natural language.

3.1. Collection of novice users' physical-computing description

In the current work, we identify art and design undergraduate students as our target user group, as they may have limited experience in technical development but could be often motivated to prototype their art/design ideas. We analyse 152 student-project reports from an Arduino-based physical-computing course which has been offered in the past three years in a local university. The purpose of this analysis is to understand how users with beginner/intermediate levels of knowledge on coding and circuit prototyping may describe their physical-computing ideas, and how the description would be mapped to electronic components and circuits. This course is offered to the second-year undergraduate students of interactive art and design, with the prerequisite knowledge of basic computer programming. In the end of the course, the students need to finish a creative physical computing project using Arduino (Arduino, 2019). They need to answer two main questions in details: *"What they build"* and *"How they do it"*, in their project reports. Under the *"What they build"* section, the student needs to generally describe his/her idea, and describe the interactive features. Under the section *"How they do it"*, the student needs to report the technical details of the system implementation, include what electronic components they use, how they work, the circuit schematic, and the code.

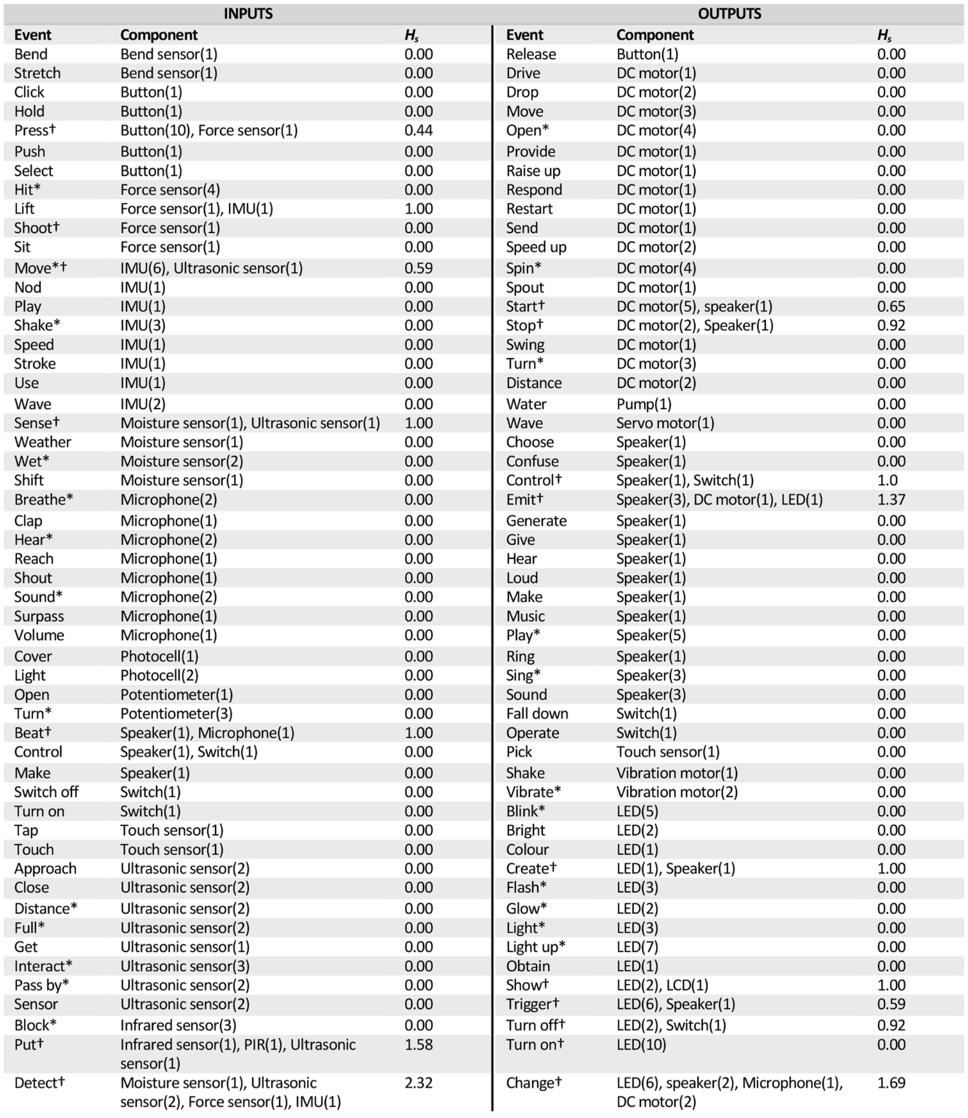
We manually went through all the reports, and observed that most of the students tend to describe their system ideas in a form of logical causal connection (i.e. input causes output). Among all the student reports, 126 reports describe their systems using the cause-effect sentences. For instance, a student describes his system as "*When player lift the dumbbell five times, one of the LED would be turned on, and while every single LED light up, theres a ringtone to remind you*", and another students describes "*When we open the cover on the top of the box and let light in, it will trigger the light*". For the rest of the student reports, ten described their system functions directly using the object names, such as "piano" (7) and "guitar" (3); nine reports described their output as video or animation on the computer screen, instead of the output in a physical-computing system; nine systems were described in a too general manner, such as "*The light reacts to user's different input*". The aforementioned non-causal system description are out of the focus of our current paper. In addition, exiting works tend to describe and construct the mathematical models of the controlling systems (e.g., robots) based on the causality mapping between their input and output (Boland, 1979; Hart et al., 1972; Iwakiri and Matsumoto, 2011). To this end, we focus on the cause-effect sentences for constructing our database. In total we extract 154 cause-effect sentences that are used to describe the features of the physical-computing systems. We categorize these sentences into in the four common patterns shown in Table 1. All these sentences consist of causal relation, including one or multiple pairs of cause and effect events. Therefore, we consider that while describing their ideas, most students tend to describe the system as a black box which will take the input event as the cause, and provide the output event as the effect. We construct a linguistic model for describing a physical-computing system as *"a causal sentence encoding the input and the output events"* as follow. For these 154 sentences, we invited an English teacher to label the cause terms (i.e. input) as "*<e1 >... </e1>*" and the effect terms (i.e. output) as "*<e2>... </e2>*". For instance, in Case 1 in Table 1, the student wants to make a baton to control a toy bird singing. From his/her description, it is a system that takes the *"swing"* event as input, causing a *"sing"* event. To keep the consistency for later system implementation and reduce the ambiguity of multiple events, we label the verb phrases as the input/output events prior to the nouns in a sentence. Based on our analysis, some systems consist of multiple inputs and outputs. In Case 2 in Table 1, the student designs a toy stick reacting to different inputs. In this case, both *"press"* and *"wave"* are denoted as the input events while both *"played"* and *"blinked"* are the output events. Case 4 shows an example of one input, *"opening"*, corresponding to multiple outputs, *"turn on", "turning around"*, and *"plays"*.

3.2. Lexical circuit-event database

Based on the causal sentences extracted from students' reports, we construct a lexical circuit-event database, to study the mapping between words and electronic components. For each project report, we manually extract the used electronic components. For example in Case 1 in Table 1, the student uses an Inertial Measurement Unit (IMU) to detect the "swing" movement, and a buzzer to play the music. Therefore, we can create a semantic correlation for the input event "swing" with the component "IMU", and the output event "sing" with the output component "speaker". Though we mostly label the verb phrases as the events prior to the nouns to reduce ambiguity, ambiguity may still occur occasionally. Some events and verb phrases could be general, and may correlate with multiple components. For instance, the correlated component for the event/word "detect" highly depends on its object. In Case 3 mentioned above, the student wants to make an automatic irrigation system that can automatically water the plant if there is not enough moisture in the soil. From his/her description, we label "detect" and "release" as the input and the output events respectively, while "detect" and "release" can be related to several electronic components. In this case, as the student implemented, the component for "release" is "water pump" according to its object "water", and the component for "detect" is "humidity sensor" according to its object "humidity". To identify the ambiguity, we compute the Shannon Entropy (Shannon, 1948) for each event as shown in Eq. (1).

Table 1
Examples of cause-effect sentence from the reports.

Categories	Count	Example	
One input to one output	108	*When the user <e1>swing</e1> the baton, a bird will <e2>sing</e2> a song.*	Case 1
Multiple input to multiple outputs	9	*When we <e1>press</e1> the button on the handle and <e1>wave</e1> the stick at the same time, music will be <e2>played</e2> and the LED light bulb will be <e2>blinked</e2> according to the rhythm of the music.*	Case 2
Multiple input to one output	9	*Out invention can <e1>release</e1> water regularly to water the plant and <e1>make</e1>. a sound once the moisture sensor <e2>detect</e2> the humidity of the plant reached the point we have set.*	Case 3
One input to multiple outputs	28	*After <e1>opening</e1> the music box, the lights inside the box <e2>turn on</e2>, the barbie doll head is <e2>turning around</e2> and the horror music <e2>plays</e2>.*	Case 4



INPUTS			OUTPUTS		
Event	Component	H_s	Event	Component	H_s
Bend	Bend sensor(1)	0.00	Release	Button(1)	0.00
Stretch	Bend sensor(1)	0.00	Drive	DC motor(1)	0.00
Click	Button(1)	0.00	Drop	DC motor(2)	0.00
Hold	Button(1)	0.00	Move	DC motor(3)	0.00
Press†	Button(10), Force sensor(1)	0.44	Open*	DC motor(4)	0.00
Push	Button(1)	0.00	Provide	DC motor(1)	0.00
Select	Button(1)	0.00	Raise up	DC motor(1)	0.00
Hit*	Force sensor(4)	0.00	Respond	DC motor(1)	0.00
Lift	Force sensor(1), IMU(1)	1.00	Restart	DC motor(1)	0.00
Shoot†	Force sensor(1)	0.00	Send	DC motor(1)	0.00
Sit	Force sensor(1)	0.00	Speed up	DC motor(2)	0.00
Move*†	IMU(6), Ultrasonic sensor(1)	0.59	Spin*	DC motor(4)	0.00
Nod	IMU(1)	0.00	Spout	DC motor(1)	0.00
Play	IMU(1)	0.00	Start†	DC motor(5), speaker(1)	0.65
Shake*	IMU(3)	0.00	Stop†	DC motor(2), Speaker(1)	0.92
Speed	IMU(1)	0.00	Swing	DC motor(1)	0.00
Stroke	IMU(1)	0.00	Turn*	DC motor(3)	0.00
Use	IMU(1)	0.00	Distance	DC motor(2)	0.00
Wave	IMU(2)	0.00	Water	Pump(1)	0.00
Sense†	Moisture sensor(1), Ultrasonic sensor(1)	1.00	Wave	Servo motor(1)	0.00
Weather	Moisture sensor(1)	0.00	Choose	Speaker(1)	0.00
Wet*	Moisture sensor(2)	0.00	Confuse	Speaker(1)	0.00
Shift	Moisture sensor(1)	0.00	Control†	Speaker(1), Switch(1)	1.0
Breathe*	Microphone(2)	0.00	Emit†	Speaker(3), DC motor(1), LED(1)	1.37
Clap	Microphone(1)	0.00	Generate	Speaker(1)	0.00
Hear*	Microphone(2)	0.00	Give	Speaker(1)	0.00
Reach	Microphone(1)	0.00	Hear	Speaker(1)	0.00
Shout	Microphone(1)	0.00	Loud	Speaker(1)	0.00
Sound*	Microphone(2)	0.00	Make	Speaker(1)	0.00
Surpass	Microphone(1)	0.00	Music	Speaker(1)	0.00
Volume	Microphone(1)	0.00	Play*	Speaker(5)	0.00
Cover	Photocell(1)	0.00	Ring	Speaker(1)	0.00
Light	Photocell(2)	0.00	Sing*	Speaker(3)	0.00
Open	Potentiometer(1)	0.00	Sound	Speaker(3)	0.00
Turn*	Potentiometer(3)	0.00	Fall down	Switch(1)	0.00
Beat†	Speaker(1), Microphone(1)	1.00	Operate	Switch(1)	0.00
Control	Speaker(1), Switch(1)	0.00	Pick	Touch sensor(1)	0.00
Make	Speaker(1)	0.00	Shake	Vibration motor(1)	0.00
Switch off	Switch(1)	0.00	Vibrate*	Vibration motor(2)	0.00
Turn on	Switch(1)	0.00	Blink*	LED(5)	0.00
Tap	Touch sensor(1)	0.00	Bright	LED(2)	0.00
Touch	Touch sensor(1)	0.00	Colour	LED(1)	0.00
Approach	Ultrasonic sensor(2)	0.00	Create†	LED(1), Speaker(1)	1.00
Close	Ultrasonic sensor(2)	0.00	Flash*	LED(3)	0.00
Distance*	Ultrasonic sensor(2)	0.00	Glow*	LED(2)	0.00
Full*	Ultrasonic sensor(2)	0.00	Light*	LED(3)	0.00
Get	Ultrasonic sensor(1)	0.00	Light up*	LED(7)	0.00
Interact*	Ultrasonic sensor(3)	0.00	Obtain	LED(1)	0.00
Pass by*	Ultrasonic sensor(2)	0.00	Show†	LED(2), LCD(1)	1.00
Sensor	Ultrasonic sensor(2)	0.00	Trigger†	LED(6), Speaker(1)	0.59
Block*	Infrared sensor(3)	0.00	Turn off†	LED(2), Switch(1)	0.92
Put†	Infrared sensor(1), PIR(1), Ultrasonic sensor(1)	1.58	Turn on†	LED(10)	0.00
Detect†	Moisture sensor(1), Ultrasonic sensor(2), Force sensor(1), IMU(1)	2.32	Change†	LED(6), speaker(2), Microphone(1), DC motor(2)	1.69

Fig. 1. All event phrases extracted from 126 student project reports which use causal sentences for project description. The number in bracket indicates the component's usage frequency. Star symbol (*) highlights the merged event. Dagger symbol (†) denotes the ambiguous events.

$$H_s = \sum_{i=1}^{n} p_i I_i = -\sum_{i=1}^{n} p_i \log_2 p_i \quad (1)$$

where p_i is the probability of a component to occur under event i, n is the total number of components. I is the information function of event i with probability p_i, which is defined as $\log_2 p_i$ by Shannon (1948). The entropy values for all the labeled events are shown in Fig. 1. $H_s = 0$ indicates the events that are unambiguous. The higher the entropy score is, the more ambiguous it is. We treat an event with a entropy score lager than 0.5 as the ambiguous event for additional processing in our algorithm which will be described later in Section 5.2. In total we found 74 unique phrases for the input events and 72 unique phrases the output events. To regularize the event, we apply a merging scheme for the event phrases that have similar meaning. For example, we merge "move" and "movement", "spin" and "spinning", and so on. This reduces the number of events to 53 for both input events and output events (Fig. 1). We also summarize 11 input electronic components and 8 output electronic components that commonly used in the students' reports (Fig. 2). This lexical database serves as the foundation of the FritzBot implementation, including the training of the machine-learning model for natural language process and the solution-generation mechanism which we will describe in details later.

3.3. Design features of FritzBot

We design FritzBot as a conversational agent that aims to facilitate circuit construction and coding for novice users. Considering the problems that may be encountered by novice users during a physical-computing task (Mellisy et al., 2016; Waterhouse, 2016), such as component selection, wiring, and programming, we identify a list of design features that need to be reflected in FritzBot.

Solution recommendation Previous works (Mellisy et al., 2016; Waterhouse, 2016) suggest that users mostly encounter the problems of selecting correct components, designing circuit, and composing the microcontroller programs. Therefore, we design FritzBot to generate the schematic of a Arduino-based circuit and code with the input and the output components based on the causal relationship extracted from the user's input sentence. Meanwhile, FritzBot will also generate the textual responses with a pre-defined format to explain the generated solution.

In addition, we support error correction. If users' input contains a specific component (e.g *"if the distance detected by the ultrasonic sensor... "*), FritzBot will detect the component phrase in the sentence and check the correspondence between the user-mentioned sensor and the extracted event by checking the lexical circuit-event database. If user mentioned the wrong component (e.g. *"if the distance detected by the infrared sensor... "*), FritzBot will retrieve and suggest the correct one (*"ultrasonic sensor"*) to achieve the desired goal (*"detect the distance"*).

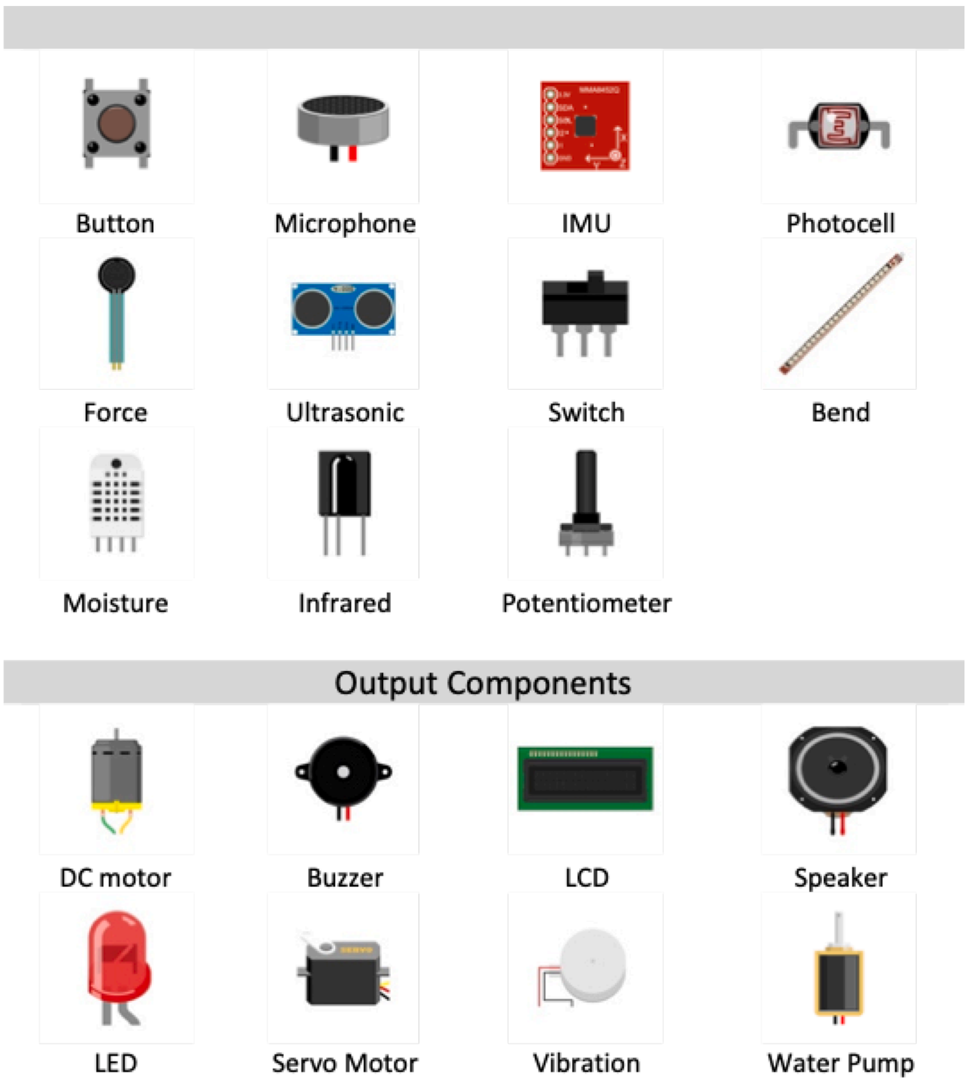


Fig. 2. Circuit components that are commonly used by the students.

Event customization FritzBot allows users to customize the levels/parameters of the detected input/output events. For a detected event, if there is no adverb or adjective detected in the user-input sentence, FritzBot will ask user to textually parameterize the event through text response. In FritzBot, we allow users to define the event attributes for each event (e.g. adjust the brightness for a light as *"very bright"* or *"dim"*, define a distance threshold as *"closed"* or *"far"*, etc.)

Step-by-step construction guidelines Another problem commonly encountered by users is mis-wiring (Waterhouse, 2016). We design FritzBot to generate the wiring guidelines in the format of multiple steps. We also allow users to track back to any step if it is necessary.

Solution restoration FritzBot allows the user to restore to the previous generated solution if he/she is not satisfied with the current one.

Components dictionary For the educational purpose, we design FritzBot to support natural-language question/query-based interaction on the usage of components. FritzBot is able to take questions like *"Can you tell me how to detect the distance?"*. It will search from the database, retrieve the component *"ultrasonic sensor"* correlated to the detected event *"distance"*, and generate a template circuit and code of that particular component. If a user is curious about how to use a component, he/she could select that component's icon in the Fritzing interface, and enter his/her question (e.g. *"How to use it"*). FritzBot will provide the template circuit and code, with predefined textual responses.

The implementation of the features above will be described in details in Section 5.

4. System walk-through

In this section, we will demonstrate the main functions of FritzBot with an running example. Alan is a bachelor student majored in design. He would like to build a smart robot that will react to different human activities as follow: 1) the robot will sing a song loudly when there is someone getting close to it; 2) it will wave its hand and blink its eyes while being touched; 3) it will run away if someone is shouting at it. After formulating his design idea, Alan starts to create a physical-computing prototype with FritzBot.

Once Alan enters that he would like to make a robot, FritzBot replies him to ask for more details about the robot (Fig. 3a). To describe the first function, Alan enters *"The robot will sing a song when we get close to it"*. After a few seconds of data processing, FritzBot generates a semi-completed system shown in the canvas. In addition, FritzBot asks Alan for more detailed parameters for the "sing" event (Fig. 3a). Alan responds with *"I want the robot to sing loudly"*, and FritzBot refreshes the generated system in the canvas (Fig. 3b). Then Alan starts to work on the

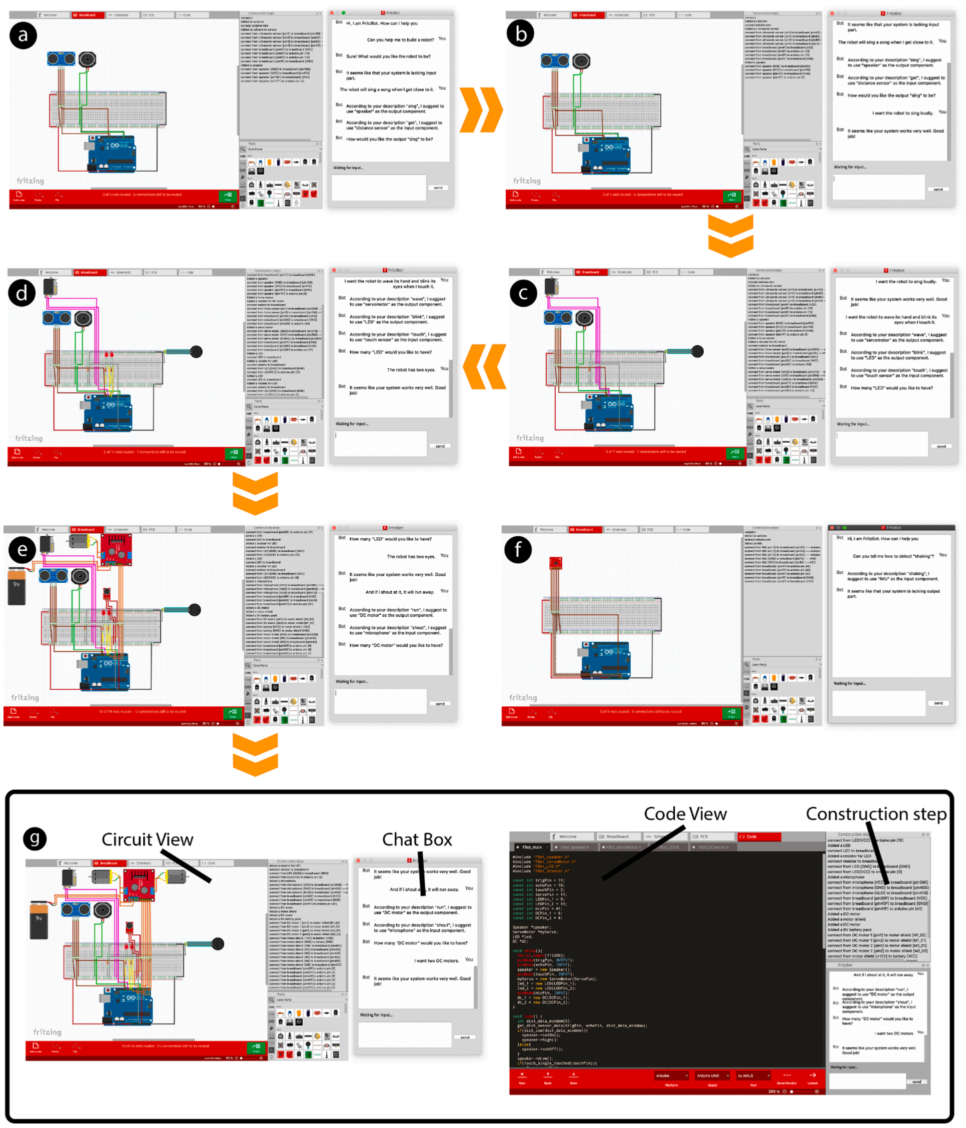


Fig. 3. System walk-through.

second function, by telling FritzBot *"I want the robot to wave its hand and blink its eyes when I touch it"*. FritzBot suggests to use a servo motor and LED for the *wave* event and the *blink* event respectively. FritzBot also asks Alan how many LED he would like (Fig. 3c). Alan replies with *"The robot has two eyes"*. FritzBot then responds with a system containing two LEDs (Fig. 3d).

Lastly, Alan enters *"And if I shout at it, it will run away"*. FritzBot tells Alan that he should use a microphone to detect the *"shout at"* event and use DC motors for *"run away"*. FritzBot then asks Alan for the quantity of DC motor (Fig. 3e). Alan decides that the robot should have two wheels, and enters *"I want two DC motors"*. FritzBot replies with a completed system (Fig. 3g). Alan then opens the construction-steps window, and follows the instructions to build the circuit step by step (Fig. 3g top-middle). He can track back to any previous step for a clearer view of wires. After finishing the circuit, he then connects the Arduino UNO to his laptop, navigates to the code view, and uploads the code (Fig. 3g bottom).

After prototyping the robot, Alan wants to add one more function to the robot. He would like the robot to know if there is people shaking its hand. He is trying to do that by himself, but he don't know how to detect the "shaking" action. He asks FritzBot: *"Can you tell me how to detect 'shaking'?"*. FritzBot replies that an IMU is suggested for detecting "shaking" and shows a sample circuit (Fig. 3f). Alan is satisfied with the response, and starts to explore the physical-computing system by himself.

While existing physical-computing prototyping platforms (e.g. Circuito.io, 2019) could also provide the wiring solutions for the electrical components used in the example above, Alan may still encounter difficulties on deciding what components he should use at the very beginning. For example, Alan may not know which sensor to be used to detect the event of "getting close to the robot". Thus, he may need to search on the Internet or go through the course materials. After finding the correct sensor to use, he need to again check the textbook or online example projects for the sample code to drive the sensor, and integrate the sample code into his own project. Alan needs to repeat this process of searching and integrating for all the other functions of his idea, and this whole process could be time-consuming. Using FritzBot, he could ask high-level questions specifically related to the project, and save time for the creative ideation process.

5. System Implementation

The current prototype of FritzBot is implemented as a plug-in of Fritzing (Knörig et al., 2009). Fig. 4 depicts the system overview of FritzBot. FritzBot consists of three parts: (1) A neural network to extract the cause/input and the effect/output events from a sentence input by the user; (2) A back-end system to process the extracted events and retrieve the corresponding components; and (3) a front-end user interface for receiving inputs and showing outputs.

5.1. Named-entity recognition

The core feature of FritzBot is to extract the input event and output event in a sentence for details. For this purpose, we train a BiLSTM-CRF-based neural network for named-entity recognition based on our dataset.

Dataset preparation As discussed before, we invited an English teacher to label the cause and the effect terms in the 154 sentences collected from the student reports. The teacher is instructed to select verbs prior to nouns to represent the cause/effect events. We do this mainly because of our observation that in most cases, a verb could better represent the event with a lower ambiguity compared to a noun, as shown in the entropy values in Fig. 1. However, a few cases may only contain nouns to represent the events. For instance, for a sentence like *"The light and music will become weird and spooky when another sensor has sensed someone passes through."*, we instruct the English teacher to label the noun *"light"* and *"music"* for the output events and the verb *"passes through"* for the input event. After labeling, we invited two undergraduate students majored in English to rephrase the sentence for data augmentation, expanding the dataset to 634 sentences.

BiLSTM-CRF model In order to label the input and the output entities from a sentence which is modeled as a list of sequential data (i.e., a sequence of words), we implement a named-entity recognition algorithm for sentence processing. Similar to the existing works (Huang et al., 2015; Lample et al., 2016), we build a BiLSTM-CRF model to label entities in a sequence of words (i.e. a sentence). In FritzBot, We implement the BiLSTM model as follow:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2}$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \tag{3}$$

$$c_t = f_tc_{t-1} + i_t\tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{4}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{5}$$

$$h_t = o_t\tanh(c_t) \tag{6}$$

where σ is the sigmoid function, i, f, c and o are input gate, forget gate, cell state, and output gate respectively. The weight matrix subscripts have the meaning as the name indicates. For example, W_{hi} denote the

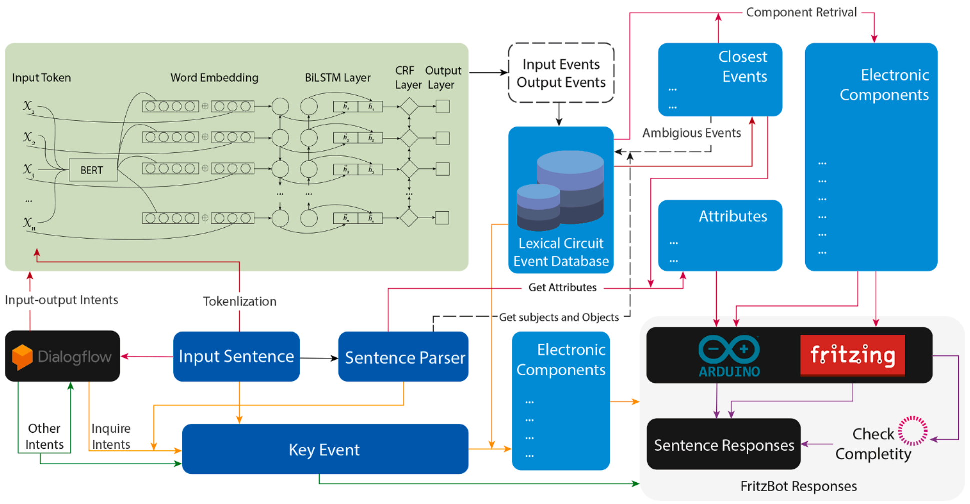


Fig. 4. FritzBot system overview. The input sentence was first uploaded to the Dialogflow server for intent recognition. If the sentence is describing a causal logic (red arrow), it will be pass to a BiLSTM-CRF network (green block) for input and output events recognition before retrieving components from the lexical circuit database and the attributes recognition process (light blue block). FritzBot will then check current system completeness and generate circuit preview, code, and responses (gray block). Yellow arrow indicates the workflow if the user is asking FritBot for component usage or component selection. If the user is answering FritzBot's question, it will be recognized as other intents (green arrow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

weight matrix for computing input gate contributing from the hidden vector. W_{xf} denotes the weight matrix for computing forget gate contributing from x, etc. b denotes the bias for each gate. h_t is the hidden vector that output at each time step and passing to the next LSTM cell. Instead of using a single direction LSTM, we implement a bi-directional LSTM (BiLSTM) (Schuster and Paliwal, 1997), which have been proved to perform better in sentence named-entity recognition task (Huang et al., 2015). For a given sentence $X = (x_1, x_2, x_3 ... x_n)$ containing a sequence of n words, each word is represented as a d-dimensional feature vector. We compute the feature vector by concatenating the vector of Part-of-Speech (POS) tags (Tomberlin, 2003; Toutanova et al., 2003) and a word vector computed by a pre-trained BERT model (Devlin et al., 2018a). Specifically, we generate the POS tags using the NLTK toolkit (Loper and Bird, 2002), in the format of a 35-dimensional vector. The pre-trained BERT model outputs a word-embedding vector of length 512. Therefore, the whole feature vector is 547 dimensional (i.e., $d = 547$). The BiLSTM computes each word vector sequentially from left to right and output a representation feature vector $\overrightarrow{h}_t$ for word x_t. Since the information in a sentence from both past (previous few words) and future (next few words) are both useful for sequential tagging, we compute another feature vector $\overleftarrow{h}_t$ from right to left of the sentence for word x_t. The final output feature vector of the BiLSTM at word x_t would be the concatenation of the left and the right representation $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$.

The input and the output events/entities in a sentence is highly related to their neighbor words (i.e. when, if, before, etc.). To encode the neighbor tag feature in predicting the current tag, we implement a CRF model (Lafferty et al., 2001) on top of the BiLSTM output to select the best predicted sequence. We follow the same pipeline proposed by Lample et al. (2016) which was proved to perform well on named-entity recognition task. For an input sequence to the network $X = (x_1, x_2, x_3 \dots x_n)$ and a sequence of prediction $Y = (y_1, y_2, y_3 \dots y_n)$, we define its score to be:

$$\mathrm{score}(X, Y) = \sum_{i=0}^{n} A_{y_i y_{i+1}} + \sum_{i=1}^{n} h_{i, y_i} \tag{7}$$

where A is the transition matrix. Specifically, A_{ij} represent the possibility of the transition from tag i to tag j. A is a trainable parameter in the CRF model. During training, we initialize it as an all-zero matrix. We compute the SoftMax possibility of the target sequence Y as follow:

$$P(Y|X) = \frac{e^{score(X,Y)}}{\sum_{\widetilde{Y} \in Y_X} e^{score\left(X, \widetilde{Y}\right)}} \tag{8}$$

where Y_X represents all possible sequences. $\widetilde{Y}$ denotes every individual sequence during the iteration. During training, we maximize the log-likelihood of the ground truth tag sequence:

$$\log(P(Y|X)) = \mathrm{score}(X, Y) - \log\left(\sum_{\widetilde{Y} \in Y_X} e^{\mathrm{score}\left(X, \widetilde{Y}\right)}\right) \tag{9}$$

While predicting, we choose a output sequence that have a maximum score computed by:

$$Y^* = \underset{\widetilde{Y} \in Y_X}{\mathrm{argmax}} s\left(X, \widetilde{Y}\right) \tag{10}$$

The aforementioned algorithm was implemented using Python 3.7.2 with TensorFlow 1.13.1 framework.

Training We split our dataset into the training set (507 sentences), the validation set (64 sentences), and the test set (63 sentences). We train our model on a desktop with one GTX 1080 Ti NVIDIA GPU with 32GB memory and one Intel i7-8700 CPU. We use a Adam optimiser (Kingma and Ba, 2014) ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with learning rate of 10^{-5} to optimise the model. During training, we apply the dropout technique (Srivastava et al., 2014) with dropout rate 0.5 to avoid over-fitting.

Performance We conducted an ablation evaluation on the performance of our BiLSTM-CRF model. Table 2 shows the performance under different hyper-parameters on the test set, where the highest accuracy occurs when using a 4-layers BiLSTM-CRF model with LSTM block in size 128. Furthermore, we compared the chosen BiLSTM-CRF structure with the BERT model (Devlin et al., 2018b) which is commonly applied for natural language processing, including named-entity recognition (Li et al., 2020). We configured the BERT model with the similar hyper-parameter (i.e. maximum length of sequence) as the chosen BiLSTM-CRF structure. The comparison result suggested that our BiLSTM-CRF model could achieve a better testing accuracy upon our dataset (BiLSTM-CRF: 92.46% vs. BERT: 89.57%).

5.2. *Circuit and code generation*

For each extracted phrase from the user-input sentences, FritzBot retrieves the most similar phrase (measured by the word distance of WordNet Miller, 1998) from the lexical circuit-event database and obtains the corresponding electronic components. For ambiguous events, the system checks the similarity between its object and the phrases in the database until the most similar and the least ambiguous event is found. That is, if an ambiguous event phrase is identified in the user's input sentence, the system will first sort the event phrases in the lexical database based on their word distances towards the object of the ambiguous phrase from smallest to largest, then the system will check the sorted list from the beginning to find the first phrase with the Shannon Entropy smaller than 0.5 (i.e. non-ambiguous event) as the final identified event for component retrieval. For each event, the system also checks its dependent adverb and adjective by using the algorithm of Part-of-Speech (POS) Tagging (Toutanova et al., 2003), to obtain the attribute of that event. If the system couldn't find any adverb or adjective for the event attributes in the input sentence, it will respond with a question to ask the user to suggest an attribute for the event. Given the retrieved electronic components, the system generates the corresponding circuit with a optimized A^* search algorithm (Hart et al., 1968) to manage Arudino pins, dynamically distributes and wires all electronic components on the canvas to reduce overlapping. The wiring rule for each electronic component follows its tutorial in the physical-computing course which we create the lexical database upon. Each wiring connection is stored as an individual element in a stack structure for the function of step-by-step construction guideline.

The code is generated in a object-oriented manner based on the electronic components and their attributes. We predefined several commonly used attributes for each component according to the physical-computing course syllabus. For instance, for LED, we allow user to define its brightness in three level (i.e. low, medium, and high) with five lighting pattern (i.e. turn on/off, blink, breathe, brighter, and dimmer). All available attributes for each component are represented as a *.json* file for the code generating algorithm. All the generated circuits and code are stored in a large stack structure for potential solution restoration by the user.

5.3. *Components dictionary and error correction*

The lexical database is used as the components dictionary, to allow the user to enquire the usage of a particular electronic component using the predefined question formats (i.e., "*Can you tell me how to <perform a certain event>?*", and "*How to use it*" with a component selection in the Fritzing GUI). With the extracted event/component, FritzBot will retrieve and display the template circuit and code associated in the database.

In the reports, we observed that some students tended to describe the system function along with a specific electronic component if they have

Table 2
Performance of the BiLSTM-CRF network with different hyper-parameters.

Model			Accuracy
	Number of Layer	LSTM size	
BLSTM-CRF	1	128	91.12%
	4	**128**	**92.46%**
	8	128	90.64%
	1	256	91.12%
	4	256	91.12%
	8	256	90.67%
CRF only			91.01%

some prior knowledge. However, the components that the users provided may not always be correct. For example, the user may say "*If I use an infrared sensor to detect distance...* " which is not correct. In this case, FritzBot performs the error-correction feature with two steps. First, if a circuit component is detected in the user's input, FritzBot will first search for the corresponding event in the same input sentence based on the universal within-sentence dependencies (Nivre et al., 2016) computed by the StanfordNLP toolkit (Qi et al., 2019). Take the aforementioned sentence as example, the event "*distance*" will be detected in the user input with the component "*infrared sensor*". Second, FritzBot will then search the existence of the particular event-component mapping in the lexical database. If the mapping does not exist, FritzBot will identify that the user's input may potentially contain an error, and suggest the correct component (i.e. "*ultrasonic sensor*" for detecting "*distance*"). In this case, FritzBot will respond with a pre-defined template, "*You should use <ultrasonic sensor>instead of <infrared sensor>to detect <distance>*", and generate a corrected circuit result.

5.4. Natural-language output

While FritzBot mainly focuses on circuit and code generation for a physical-computing task, it also generates natural-language feedback in order to facilitate users' understanding of the generated circuit. There are two types of natural-language feedback: the response and the follow-up question. For the natural-language response, we predefined a set of sentence templates for different types of responses (i.e. the confirmation response and the error-correction response). The confirmation response is defined as "*According to your description <e>, I suggest to use <c>as the input/output component.*" where *e* is the extracted user input event and *c* is the corresponding retrieved electronic component. It would be triggered while FritzBot is generating the circuit solution or setting the circuit attributes (see section 5.2). The error-correction response ("*You should use <c1>instead of <c2>to detect <e>.*") is for informing the user that there may be a wrong electronic component for his/her desired event (see section 5.3), where *c1* is the correct component, *c2* is the user-inputted component, and *e* is the extracted event.

FritzBot treats a circuit that contains at least one input event and one output event along with their attributes as a complete system. FritzBot will check the completeness of the system during the user's prototyping process. If the system is lacking any event attribute, FritzBot will generate a follow-up question to ask the user for providing a corresponding attribute for the event. We predefined several follow up question templates to increase the language diversity. Specifically, FritzBot will ask "*How many <c>would you like to have?*" to get the quantity of the retrieved component *c*, or "*How would you like the input/output <e>to be?*" to ask for the attribute of the corresponding input/output event *e*, or "*It seems like that your system is missing the input/output part. Would you like to provide more information?*" if any input/output event is missing.

5.5. Front-end user interface

We implemented the front-end interface of FritzBot as a plug-in of Fritzing (Knörig et al., 2009) using C++ with Qt framework. The interface communicate with the back-end system (implemented using Python 3.7.2) through TCP/IP protocol.

6. User study

We conducted a between-subject user study to assess how novice will create a physical computing system using FritzBot, particularly to investigate if FritzBot can significantly reduce the time and the effort for physical-computing system construction.

6.1. Participants

Twenty-four participants (all design-majored students from a local university, 12 males and 12 females, averagely aging 22.1 years old with SD of 2.50) were involved in our study. The participants are randomly and evenly assigned to three groups: 1) Arduino Group (AG): using Arduino IDE only, 4 males and 4 females, Age Mean = 22.9, 2) Circuito Group (CG): using Circuito (Circuito.io, 2019) and Arduino IDE, 6 males and 2 females, Age Mean = 21.8, and 3) FritzBot Group (FG): using FritzBot, 2 males and 6 females, Age Mean = 21.1. The native Arduino IDE was chosen as the basedline. While there are existing research prototypes on facilitating physical-computing tasks, Circuito was chosen for comparison because it, as an commercial product, is widely available and used within the community. In addition, the existing research prototypes are not available with enough details for reproduction. Therefore, we adopt the evaluation strategy followed by other related works (Anderson et al., 2017; Kim et al., 2019) to compare our solution with the tools available in the market. All the three groups are provided with sufficient electronic components to finish the task, and allowed to search relevant knowledge on the Internet. We also include some distractor components in the component list to increase the level of confusion. We also provide the participant a set of digital documents on the basic introduction of electronic component and Arduino. The provided documents are compiled by a physical-computing teacher who has been teaching the related courses in a local design-related undergraduate program for five years.

Prior to the study, the participants rate their electronic experience and programming experience at the level of "Never Try" – 1, "Beginner" – 2, "Intermediate" – 3, or "Expert" – 4. As the Shapiro–Wilk test shows a significant departure of the participant's self-rated technical skills from the normal distribution, we adopted the non-parametric statistical analysis on these data. Kruskal Wallis Test indicates there was no different among the three groups in terms of electronic experience (AG: 2.13, CG: 2.25, FG: 2.00, $H(2) = 1.208$, $p = 0.69$) and programming experience (AG: 2.75, CG: 2.5, FG: 2.63, $H(2) = 5.565$, $p = 0.06$). Genderwise, Mann-Whitney U Tests showed no significant difference between females and males in terms of the technical background (Electronics: female − 1.36, male − 1.38, $Z = 0.027$, $p = 0.978$; Programming: female − 1.81, male − 1.92, $Z = 0.057$, $p = 0.919$).

6.2. Task

During the user study, users were asked to finish a “smart baton” project. The “smart baton” has the follow function: with the follow features/functions: 1) it will start vibrating while being presses; 2) a light on it will light up while being shaken; 3) it will play music loudly while being pressed and shaken at the same time. This task is derived from the physical-computing curriculum in the local university where we retrieved the student reports. Based on the physical-computing curriculum in the participants’ university, this task is of moderate difficulty for a 2nd-year design/art-majored student, and it can be completed in under an hour for novice using 5 electronic components, including 2 input components (i.e. a button and an accelerometer) and 3 output components (i.e. an LED, a vibration motor, and a speaker/buzzer) (Fig. 5).

6.3. Procedure

Before the main task, the participant was given a brief tutorial of basic electronics and Arduino, including the functions of common electronic components, the basic usage of Breadboard, and the basic logic of Arduino programming. Depending on the participant’s group, they were also given a brief instruction about FritzBot or Circuito.io or Arduino IDE.

After the instruction, the participant was given a video (Fig. 6) demonstrating all functions of the task system. The video sequentially shows three functions with each of them being repeated three times. To avoid bias, we hide all electronic components for the implementation in the video, only showing the final input and output. For output that could not be visualized (e.g. vibration), we show a icon with animation. We do not provide any textual description in the video. After watching the video, the participant needs to describe each function that he/she saw from the video to ensure that he/she clearly understood the task. The participant was given 45 min to finish the task. With his/her permission, we recorded each participant’s time spent and level of completion. We also recorded the screen operation of each participant, to estimate the time for software usage (e.g., circuit design and coding), information searching through the web browsers and the provided materials, and physical circuit implementation. After the experiment, participants are required to answer a usability questionnaire on their impressions (1 - strongly disagree, 7 - strongly agree) of the system (i.e. Arduino IDE, Circuito.io, and FritzBot) and perceived workload. Following previous work on creative toolkit evaluation (Chen et al., 2019; Kazi et al., 2011; Zhu and Zhao, 2013), we choose the questionnaire items from the USE questionnaire (Lund, 2001) for the user-experience and the usability evaluation, and the NASA-TLX questionnaire (Hart and Staveland, 1988) for the workload evaluation.

6.4. Results

In this section, we discuss the analysis on the time-completion time and the user experience across the three different groups. As the Shapiro-Wilk test shows a significant departure of the data from the normal distribution, we adopted the non-parametric statistical analysis for the participants’ task performance and their responses to the questionnaire.

6.4.1. Task-completion time and rate

In overall, Mann-Whitney U Test doesn’t reveal any significant difference between the female and the male participants in terms of the task-completion time. All FG participants successfully completed the task, while there is only one AG participant and one CG participant completed the task. Therefore, we treat the time spent by the other AG and CG participants as 45 min which is the set length of the task. The Kruskal Wallis Test shows a significant effect of the grouping on the task completion time ($H(2) = 18.27$, $p < 0.0005$). The pair-wise Mann-Whitney U Tests show that the FG participants spent significantly shorter time (Mean = 26.5 min, SD = 2.96) than the AG participants (Mean = 44.3 min, SD = 0.75, $Z = 3.40$, $p < 0.005$) and the CG participants (Mean = 44.4 min, SD = 1.65, $Z = 3.46, p < 0.005$). Fig. 7 shows the time spent on different functions of the three groups. We treat each function as a subtask to represent the observable milestones. As most AG and CG participants couldn’t complete the third functions, we mainly compare the completion time for the first two functions/subtasks. Kruskal Wallis Test shows a significant effect of the grouping on the completion time for the first and the second functions (Func.1: $H(2) = 6.36$, $p < 0.05$; Func.2: $H(2) = 16.27$, $p < 0.005$). Overall, the FG participants completed the first two functions significantly faster than the participants in the other two groups. The Kruskal Wallis Test also shows a significant difference among the groups in terms of task-completion rate ($H(2) = 13.69$, $p < 0.005$. FG: 100%, CG: 62.6%, AG: 54.1%).

We further break down the task completion time into three main parts: 1) software usage (e.g., circuit design and coding), 2) physical circuit implementation, and 3) information searching (e.g., web browsing and referring to the provided material). Specifically for the information-searching time, we focus on the time duration for a user using other tools, such as web browsers and e-book readers, to search relevant physical-computing knowledge. As we didn’t observe any FG participants switching to other software during the experiments, we set their information-searching time as 0. Fig. 8 shows the time spent on different activities of the three groups.

The Kruskal Wallis Test shows that there is a significant effect of the grouping on the time for information searching ($H(2) = 17.44$, $p < 0.0005$) and physical circuit implementation ($H(2) = 10.82$, $p < 0.005$), but not software usage ($H(2) = 3.40, p = 0.183$). As there is no FG participants switching to other software for information searching, it is obvious that the time for information searching using other tools in this group is significantly shorter than the other two groups. The pair-wise Mann-Whitney Tests also show that the FG participants spent significantly shorter time on circuit implementation (Mean = 13.2 min, SD = 7.78) than the AG participants (Mean = 24.9 min, SD = 3.05, $Z = 2.84, p < 0.005$) and the CG participants (Mean = 21.3 min, SD = 5.04, $Z = 2.10, p < 0.05$). There was no difference between AG and CG in terms of the time for information searching and circuit implementation. In addition, there was no significantly difference on the software-usage time across the three groups (AG: Mean = 11.4 min, SD = 2.75; CG: Mean = 11.4 min, SD = 5.78; FG: Mean = 14.9 min, SD = 4.20).

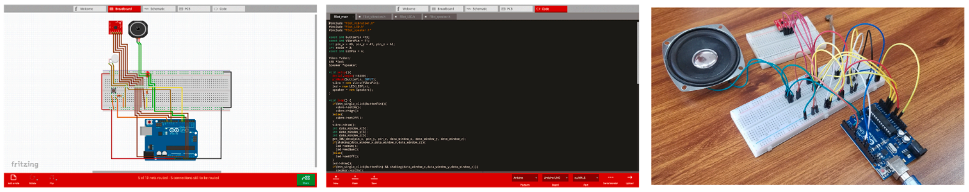


Fig. 5. User-study task. Left: The circuit plan; Middle: The code; Right: The completed physical circuit.

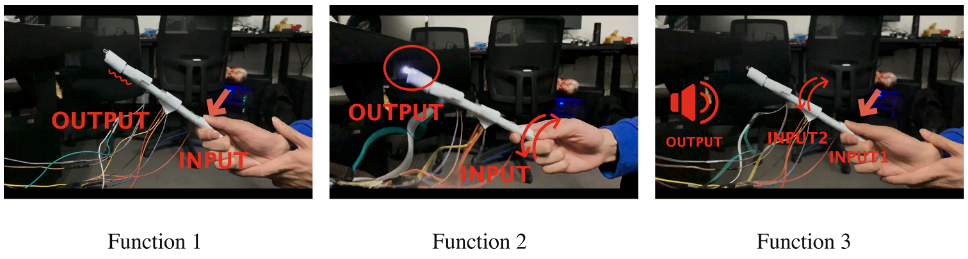


Fig. 6. Screenshot of the video demonstration, indicating three tasks for the user study.

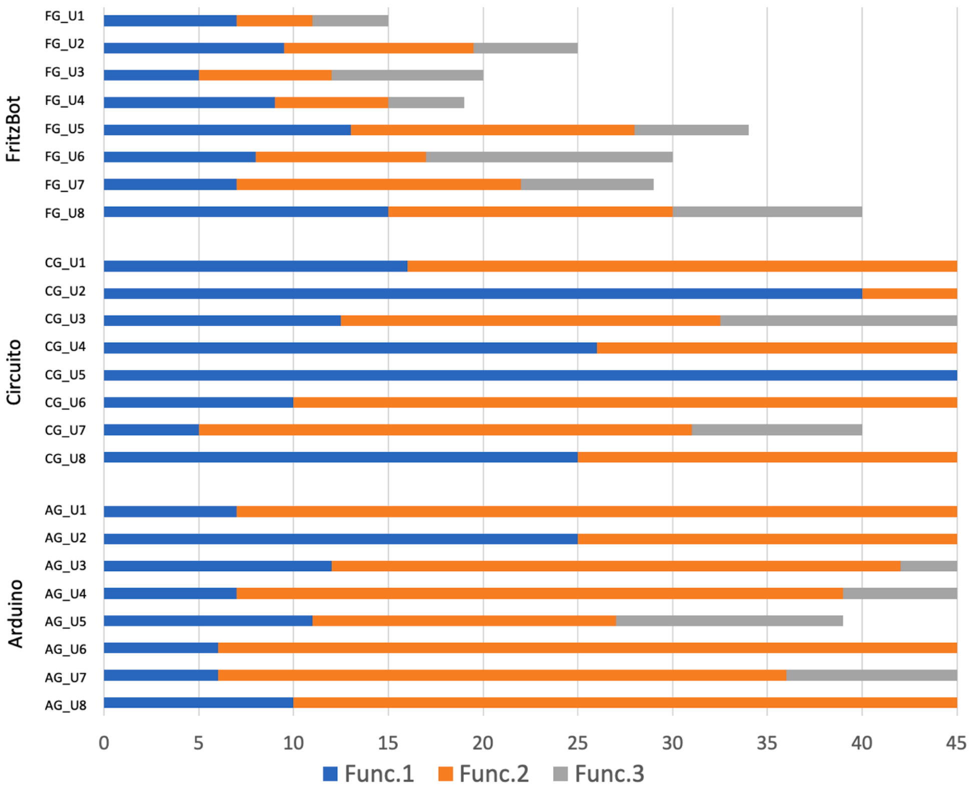


Fig. 7. Time spent on different subtasks/functions by AG, CG, and FG participants. The horizontal axis indicates the time in minutes.

6.4.2. *Usability and user engagement*

The Kruskal Wallis Tests show that the grouping condition significantly affects the participants' ratings on ease to learn ($H(2) = 10.73$, $p < 0.005$), ease to use ($H(2) = 7.13$, $p < 0.05$), ease to correct mistake ($H(2) = 6.92$, $p < 0.05$), system consistency ($H(2) = 10.11$, $p < 0.05$), usefulness on creating physical-computing system ($H(2) = 13.29$, $p < 0.005$), and helpfulness on learning physical-computing knowledge ($H(2) = 8.53$, $p < 0.05$). One user (FG_U3) told us after the experiment *"It is great to know that the accelerometer could be used to detect shaking in such a high accuracy"*. Regarding the user engagement (i.e., fun, enjoyment, and willingness to use in the future, etc.), the Kruskal Wallis Tests also show a significant difference across the three groups (Fun: $H(2) = 11.28$, $p < 0.005$; Enjoyment: $H(2) = 10.37$, $p < 0.05$; Willingness of Future Use: $H(2) = 6.12$, $p < 0.05$; Willingness of Recommendation: $H(2) = 8.22$, $p < 0.05$).

Pair-wise comparison using Mann-Whitney U Tests (see Table 3 for details) show that compared to Circuito and Arduino IDE, FritzBot is rated significantly higher score in the ease to learn, the feature integration, the system consistency, the confident of using, the support on rapid prototyping, the fun and the joy of use, and the willingness for future use. In addition, FritzBot is rated significantly easier to use and correct mistakes, more useful for rapid-prototyping and learning physical computing, and more supportive for productivity and creativity than Circuito, while there is no significant difference between FritzBot and Arduino IDE in these aspects.

6.4.3. *Perceived workload*

The average total NASA-TLX score of the experimental group is 20.88/60 (SD = 2.08), and the average total score of the controlled group is 31.63/60 (SD = 1.30). Fig. 9 shows the NASA-TLX results.

The Kruskal Wallis Tests show that the grouping condition significantly affects the user-perceived mental demand ($H(2) = 8.92$, $p < 0.05$), temporal demand ($H(2) = 9.01, p < 0.05$), effort ($H(2) = 10.9, p < 0.005$), and frustration ($H(2) = 11.4, p < 0.005$). The pair-wise Mann-Whitney U Tests show that compared to the FG participants, the CG participants rate significantly higher in terms of mental demand ($U = 7.00, p < 0.01$), temporal demand ($U = 11.50, p < 0.05$), overall effort ($U = 3.50$, $p < 0.01$), and frustration ($U = 0.50$, $p = 0.00$). Similarly, the AG participants rated higher mental demand ($U = 9.50$, $p < 0.05$), temporal demand ($U = 5, p < 0.005$), and effort ($U = 8.00$, $p < 0.05$) than to the FG participants. One AG participant commented that "It is tiring to search how to use the components and make the circuit online". As she rated herself inexperienced in physical computing, she had to follow the online information closely, and

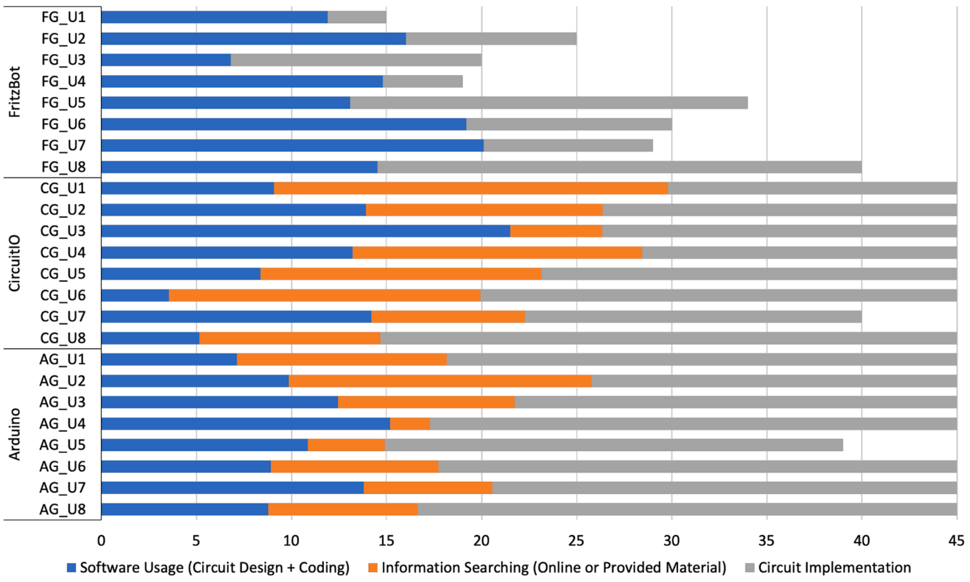


Fig. 8. Time spent on different activities by AG, CG, and FG participants. The horizontal axis indicates the time in minutes.

frequently switch between the physical circuit and the web browser to make the circuit.

The FG participants perceived better performance than the CG participants ($U = 11.50$, $p < 0.05$). There are two CG participants complaint that the circuits in Circuito are too complicated to follow. Two other CG (CG_U1 and CG_U3) participants mentioned that they could figure out the logic but it is very difficult for them to program it.

7. Discussion

Overall, our study demonstrated that with FritzBot, users with limited technical knowledge are able to finish a functional physical-computing system efficiently.

It is noted that the information-searching time for the FG participants were estimated as 0, since there was no one in this group switching to other tools for searching. This could be because the information-searching activities can be done within the FritzBot interface by chatting. This assumption echos with the result that FritzBot is rated significantly better integrated that Circuito and Arduino IDE. This also lead to slightly longer software-usage time for the FG participants than the other two groups (FG: Mean = 14.56 min, SD = 4.20; CG: Mean = 11.13 min, SD = 5.78; AG: Mean = 10.87 min, SD = 2.75). We also observed that the participants in the other two groups frequently switched across the design/coding software, the web browser, and the provided documents. The tool-switching behaviors may increase the task-completion time.

Furthermore, the circuit-implementation time for FG is significantly shorter than the other two groups (FG: Mean = 11.94 min, SD = 7.78; CG: Mean = 20.50 min, SD = 5.04; AG: Mean = 25.14 min, SD = 3.05). For the AG participants, we observe that they often switch back and forth between the process of physical circuit making and online information searching. Circuito could automatically generate the circuit design once the participant decides which component to use. The automation may reduce the need of searching how to connect a specific component, and reduce the circuit-implementation time. On the other hand, we observe that seven CG participants still conduct extensive online and document search for which component to use before generating the circuit. In contrast, we observe that all the FG participants perform a clear step-by-step pattern of task completion: Step 1) Chatting with FritzBot for solution, Step 2) Constructing the physical circuit, and Step 3) Upload the generated code, and there is very few switching behaviors back and forth across the steps.

The support of natural-language input received positive comments from the EG participants. One participant (FG_U6) commented very excitingly after she finish the first function: *"I definitely cannot wait to recommend this system to my friends!"*. There are two participants (FG_U4 & FG_U5) having similar comments on *"I will immediately purchase this system when it is available online."*. They all found it intuitive to describe their ideas in natural language because they do not need to care what components they should use and how to use them. We observe that four CG participants were struggling on choosing the accelerometer for detecting shaking. In addition, three CG participants hesitated on using vibration motor to generate vibration. All the participants search for the usage of accelerometer on the Internet. There is one CG participant (CG_U4) attempting to use DC motor to generate vibration, and he spent lots of time on searching for corresponding solutions online. In addition, FritzBot can robustly handle a variety of natural-language descriptions. During the user study, different users used different language patterns and phrases to describe a causal-effect relation and circuit events. Table 4 shows the common language patterns used by the participants and successfully processed by FritzBot.

Similar to Anderson et al. (2017), we observe that four FG participants did not use the step-by-step function to finish the circuit. For those who used this feature, two followed the generated instructions to construct the whole circuit (FG_U5 & FG_U2), and another one referred to the step-by-step descriptions to identify the pins of the accelerometer (FG_U1). Participants who used the step-by-step function commented that the this function is useful. However, there is one participant (FG_U4) found that the circuit diagram is clear enough for her to follow. She also mentioned it would be better if the step-by-step instructions could *"highlight the wires that she need to connect"*. In contrast, we observe that the CG participants found that most of the online materials do not provide a clear circuit diagram. They needed to interpret the textual instructions online or the photo of the finished circuit to construct the circuit, which is more challenging for them.

Although 7 of 8 FG participants could finish the task smoothly, FG_U6 was suffering from the error-recognition problem. His input contain some misspelling words and grammar errors, which could not be correctly recognized by FritzBot. During constructing Function#3, he did not include the description such as "at the same time" or "simultaneously", to tell FritzBot that two input events will occur together. In this case, FritzBot failed to generate the code that listens for two input pins simultaneously.

Table 3
Between-group comparison on the questionnaire ratings. The numbers within the brackets are the standard deviations. “>” indicates significant different ($p < 0.05$), and “∼” indicates no significant difference, for Mann-Whitney Test.

Questionnaire item	FritzBot mean	Arduino mean	CircuitO mean	Between-group comparison
It is easy for me to learn to use this toolkit.	6.00 (0.53)	4.00 (1.69)	3.50 (1.31)	FritzBot > Arduino ∼Circuito
I think the system was easy to use.	5.88 (0.35)	4.75 (2.25)	3.63 (1.41)	FritzBot ∼Arduino, FritzBot > Circuito
I found the various functions in this system were well integrated.	6.13 (0.35)	4.75 (1.75)	3.63 (0.74)	FritzBot > Arduino > Circuito
I thought there was too much inconsistency in this system.	2.13 (0.99)	2.88 (1.13)	3.88 (0.35)	Circuito ∼Arduino > FritzBot
I found the system very cumbersome to use.	1.75 (0.71)	3.38 (1.85)	3.50 (0.53)	Circuito ∼Arduino > FritzBot
I felt very confident using the system.	6.13 (0.64)	3.75 (1.98)	3.63 (0.92)	FritzBot > Arduino ∼Circuito
I can recover from wrong design easily with this tool.	5.50 (1.07)	4.25 (1.98)	3.50 (0.76)	FritzBot ∼Arduino, FritzBot > Circuito
I can create physical-computing systems quickly using this tool.	6.75 (0.46)	4.38 (1.06)	4.00 (1.07)	FritzBot > Arduino ∼Circuito
The tool is useful in creating physical-computing systems.	6.50 (0.53)	5.88 (0.83)	4.88 (0.64)	FritzBot ∼Arduino > Circuito
This tool is useful for helping me to learn how to create physical-computing systems.	5.75 (0.71)	5.63 (1.60)	4.13 (0.99)	FritzBot ∼Arduino > Circuito
Making physical computing with this tool is fun.	6.63 (0.74)	4.75 (1.67)	4.13 (1.46)	FritzBot > Arduino ∼Circuito
I enjoy creating physical-computing systems using this toolkit.	6.50 (0.76)	4.63 (1.69)	4.13 (1.36)	FritzBot > Arduino ∼Circuito
I became creative in physical computing with this tool.	5.25 (0.89)	4.50 (1.60)	3.75 (1.04)	FritzBot ∼Arduino, FritzBot > Circuito
I became productive in physical computing with this tool.	6.00 (1.28)	4.88 (1.19)	4.25 (0.99)	FritzBot ∼Arduino, FritzBot > Circuito
I think that I would like to use this system frequently.	5.25 (0.53)	4.00 (1.89)	3.63 (1.04)	FritzBot > Arduino ∼ Circuito

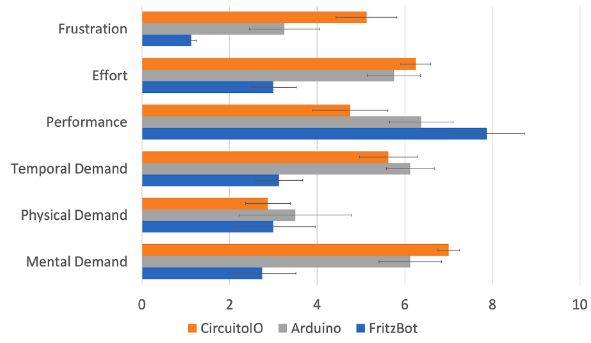


Fig. 9. NASA-TLX Results

8. Limitations and future work

During the user study, we identified several limitations for future improvement. Firstly, the performance of the BiLSTM-CRF network heavily relies on the scale of the training dataset. With our dataset derived from 152 student reports, our current prototype can correctly process most of the short causal-effect sentences. However, it sometimes fails on some complex and long sentences due to the lack of such training data, as our current dataset mainly contains short causal sentences. In addition, the current dataset only contains the textual description from one single ethic group. As users from different language background may describe their ideas differently, the current prototype may not be able to successfully analyze the input by another ethic group. The current BiLSTM-CRF model somehow relies on certain specific sentence features. For example, FG_U6’s input doesn’t include phrases such as *“at the same time”* or *“simultaneously”*, making it ambiguous for the system to correctly identify that the two input events will happen at the same time. As a proof of concept, FritzBot current support 19 electronic components (Fig. 2). The number of supported components and circuit event could be extended by adding more entities in our database, without the need of re-training the BiLSTM-CRF model. Experts in physical computing, such as instructors, can also customize and expand the lexical database by inputting their own data. We will also collect more data from larger online and offline physical-computing communities for more training data in the future, to increase the generalizability of the system. For the non-physical-computing domain that can be described using the cause-effect relationship, the FritzBot approach could be adopted by constructing the new lexical database and training the causality-prediction model with the database from that particular domain.

Secondly, we are aware of various emerging deep-learning models for natural language processing (NLP), specifically for name-entity recognition (Lothritz et al., 2020), and they may achieve better performance then the BiLSTM-CRF model which is currently used in FritzBot. To this end, the BiLSTM-CRF model is designed as one replaceable module in FritzBot which takes the user’s sentence as input, and labels the input and the output events for the later process. Therefore, the future researchers can replace this part with the emerging NLP models.

Thirdly, the FritzBot system currently can generate the physical-computing systems using Arduino only. We observed that one participant (CG_U5) finished the first function (press a button trigger a vibration) without using Arduino, which dramatically reduced the construction time and system complexity. In the future version, FritzBot will support generate circuitry without using a micro-controller if possible.

Fourth, although we currently support user customization on event attributes, all the available attribute options are predefined. For example, we allow user to define the brightness of a LED, but FritzBot will only recognizes phrases such as *bright, medium, dim*, and their synonyms. In our further version, we will provide more space for users’ customization on the corresponding components.

Last but not the least, our user studies mainly focused on the usability and the user engagement of FritzBot, without specific evaluation on the long-term educational value. However, the participants also commented on the possibility of using FritzBot in the educational settings. For instance, FG_U2 commented that “*I can learn how to use different sensors in FritzBot*”. FG_U7 said, “*I could have been more interested in physical computing if I could learn this course with FritzBot*”. Automatic content generation has been widely applied in the design of intelligent tutoring systems, such as code generation for teaching software engineering (Khmelevsky et al., 2012) and algorithm design (Gavilanes et al., 2009), circuit generation for introductory electronic courses (Beg, 2013; Macindoe et al., 2014), and so on, to enhance the student engagement in

Table 4
Participants' language usage during user study.

SENTENCE PATTERN	EVENT PHRASES FOR COMPONNETS	
	Event phrases	Electronic Components
It has a … and if I … , it will …	"press a button", "press it", "push it"	button
I want it to … when …	"shake", "wave", "swing"	IMU
When I …, then it will …	"shake", "vibrate", "rock"	Vibration motor
If I … and … at the same time, it will …	"lighten", "light up", "light"	LED
It will … when I … and … simultaneously	"play music", "sing a song", "emit sound"	Speaker

learning. For math education, Kapur suggested that intelligent problem posing with automatic solution generation prior to instruction plays a critical role in the development of conceptual understanding (Kapur, 2018). With the loosening restriction on social distancing in the post-pandemic era, we plan to deploy FritzBot to a large group of students with long-term in-class teaching and learning, to further evaluate FritzBot's educational support on physical interface and product design.

Last but not the least, our user study involved one main task/system tested with art and design students who are inexperienced in physical computing. Previous research (Lo et al., 2019) shows that expert users may have different behaviors in different physical-computing activities, compared to inexperienced users. In the future work, we plan to conduct more workshops of FritzBot, involving participants with different background, to study how FritzBot with natural-language interaction could facilitate different levels of physical-computing tasks and creative processes.

9. Conclusion

With the support of emerging machine-learning models, we present FritzBot, a conversational agent supporting novice users on creating physical-computing systems through natural-language interaction. The conversational engine of FritzBot was developed based on a BiLSTM-CRF neural network for identifying the input events and the output events phrase in users' textual description of their ideas. FritzBot can extract the causal relationship from the text, identify the input and the output components, and generate the corresponding circuit and code along with the construction guidelines. Our user study shows that compared to the circuit-autocompletion software available in the commercial market, FritzBot significantly shortens the time spent and the perceived workload for novice users on tasks of physical-computing system design and prototyping. While this initial implementation is capable to support various circuit components and behaviours, the concept that we present could be extended to support other hardware platform with more data.

CRediT authorship contribution statement

Taizhou Chen: Data curation, Investigation, Formal analysis, Software, Writing – review & editing, Visualization. **Lantian Xu:** Data curation, Software, Visualization, Writing – review & editing. **Kening Zhu:** Conceptualization, Funding acquisition, Methodology, Project administration, Writing – review & editing, Supervision.

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

Acknowledgement

This research was partially supported by the Young Scientists Scheme of the National Natural Science Foundation of China (Project No. 61907037), the Guangdong Basic and Applied Basic Research Foundation (Project No. 2021A1515011893), the Applied Research Grant (Project No. 9667189), and ACIM, School of Creative Media, City University of Hong Kong..

References

2019. Arduino - Home.
Anderson, F., Grossman, T., Fitzmaurice, G., 2017. Trigger-action-circuits: leveraging generative design to enable novices to design and build circuitry. Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, 331–342. 10.1145/3126594.3126637.
2019. BeagleBoard.
2019. Bixby | Apps - The Official Samsung Galaxy Site.
Beg, A., 2013. Automatic generation of characterization circuits-an application in academia. 2013 IEEE Frontiers in Education Conference (FIE). IEEE, pp. 661–664.
Boland Jr., R.J., 1979. Control, causality and information system requirements. Account. Organ. Soc. 4 (4), 259–272.
2019. Circuit Design App for Makers- circuito.io.
Chen, T., Wu, Y.-S., Zhu, K., 2019. Duprobo: interactive robotic autocompletion of physical block-based repetitive structure. IFIP Conference on Human-Computer Interaction. Springer, pp. 475–495.
Coelho, J., Duarte, C., Biswas, P., Langdon, P., 2011. Developing accessible TV applications. ASSETS'11: Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 131–138. https://doi.org/10.1145/2049536.2049561.
Cohen, P.R., 1992. The role of naturial language in a multimodal interface. Proceedings of the 5th annual ACM symposium on User interface software and technology UIST '92, pp. 143–149.
Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2018a. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Mlm). arXiv:1810.04805.
Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2018b. Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
Fernando, O.N.N., Cheok, A.D., Ranasinghe, N., Zhu, K., Edirisinghe, C., Cao, Y.Y., 2009. Poetry mix-up: a poetry generating system for cultural communication. Proceedings of the International Conference on Advances in Computer Enterntainment Technology. ACM, pp. 396–399.
Gavilanes, A., Martín, P.J., Torres, R., 2009. A tool for automatic code generation from schemas. International Conference on Computational Science. Springer, pp. 63–73.
Ha, D., Eck, D., 2017. A neural representation of sketch drawings. arXiv:1704.03477.
Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. 4 (2), 100–107.
Hart, P.E., Nilsson, N.J., Robinson, A.E., 1972. A Causality Representation for Enriched Robot Task Domains. Technical Report. Stanford Research Inst Menlo Park CA.
Hart, S. G., Staveland, L. E., 1988. Development of nasa-tlx (task load index): Results of empirical and theoretical research.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9 (8), 1735–1780.
Huang, Y., Kaufmann, M., Aksan, E., Black, M.J., Hilliges, O., Pons-Moll, G., 2018. Deep inertial poser: learning to reconstruct human pose from sparse inertial measurements in real time. ACM Trans. Graph. 37 (6) https://doi.org/10.1145/3272127.3275108.
Huang, Z., Xu, W., Yu, K., 2015. Bidirectional LSTM-CRF Models for Sequence Tagginga rXiv:1508.01991.
Iwakiri, S., Matsumoto, M., 2011. Investigation of robot behavior model to build causality after events. 2011 IEEE/SICE International Symposium on System Integration (SII). IEEE, pp. 1–5.

Kapur, M., 2018. Examining the preparatory effects of problem generation and solution generation on learning from instruction. Instr. Sci. 46 (1), 61–76.

Kazi, R.H., Chua, K.C., Zhao, S., Davis, R., Low, K.-L., 2011. Sandcanvas: a multi-touch art medium inspired by sand animation. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1283–1292.

Khmelevsky, Y., Hains, G., Li, C., 2012. Automatic code generation within student's software engineering projects. Proceedings of the Seventeenth Western Canadian Conference on Computing Education. Association for Computing Machinery, New York, NY, USA, pp. 29–33. https://doi.org/10.1145/2247569.2247578.

Kim, Y., Choi, Y., Kang, D., Lee, M., Nam, T.-J., Bianchi, A., 2019. HeyTeddy: conversational test-driven development for physical computing. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 3 (4), 1–21. https://doi.org/10.1145/3369838.

Kingma, D. P., Ba, J., 2014. Adam: a method for stochastic optimization. arXiv: 1412.6980.

Knörig, A., Wettach, R., Cohen, J., 2009. Fritzing a tool for advancing electronic prototyping for designers, 351. 10.1145/1517664.1517735.

Kuhn, T., 2014. A survey and classification of controlled natural languages. Comput. Linguist. 40 (1), 121–170.

Lafferty, J., McCallum, A., Pereira, F.C.N., 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning, 8, pp. 282–289. https://doi.org/10.1038/nprot.2006.61. arXiv:1011.4088v1.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C., 2016. Neural architectures for named entity recognition. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 260–270. https://doi.org/10.18653/v1/N16-1030. arXiv:1603.01360v3.

Laput, G., Dontcheva, M., Wilensky, G., Chang, W., Agarwala, A., Linder, J., Adar, E., Street, T., Street, S. S., Francisco, S., States, U., Arbor, A., States, U., 2013. PixelTone: A Multimodal Interface for Image Editing.

Lau, T., Cerruti, J., Manzato, G., Bengualid, M., Bigham, J.P., Nichols, J., 2010. A conversational interface to web automation. UIST 2010 - 23rd ACM Symposium on User Interface Software and Technology, pp. 229–238. https://doi.org/10.1145/1866029.1866067.

Li, T.J.-J., Azaria, A., Myers, B.A., 2017. Sugilite: creating multimodal smartphone automation by demonstration. Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 6038–6049.

Li, T.J.-J., Labutov, I., Li, X.N., Zhang, X., Shi, W., Ding, W., Mitchell, T.M., Myers, B.A., 2018. Appinite: A multi-modal interface for specifying data descriptions in programming by demonstration using natural language instructions. 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, pp. 105–114.

Li, T.J.-J., Radensky, M., Jia, J., Singarajah, K., Mitchell, T.M., Myers, B.A., 2019. Pumice: a multi-modal agent that learns concepts and conditionals from natural language and demonstrations. Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, pp. 577–589.

Li, X., Zhang, H., Zhou, X.-H., 2020. Chinese clinical named entity recognition with variant neural structures based on bert methods. J. Biomed. Inform. 107, 103422.

Lin, S.C., Hsu, C.H., Talamonti, W., Zhang, Y., Oney, S., Mars, J., Tang, L., 2018. ADASA: a conversational in-vehicle digital assistant for advanced driver assistance features. UIST 2018 - Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, pp. 531–542. https://doi.org/10.1145/3242587.3242593.

Lo, J.-Y., Huang, D.-Y., Kuo, T.-S., Sun, C.-K., Gong, J., Seyed, T., Yang, X.-D., Chen, B.-Y., 2019. AutoFritz: autocomplete for prototyping virtual breadboard circuits. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19, pp. 1–13. https://doi.org/10.1145/3290605.3300633.

Loper, E., Bird, S., 2002. Nltk: the natural language toolkit. In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics.

Lothritz, C., Allix, K., Veiber, L., Klein, J., Bissyande, T.F.D.A., 2020. Evaluating pretrained transformer-based models on the task of fine-grained named entity recognition. Proceedings of the 28th International Conference on Computational Linguistics, pp. 3750–3760.

Lund, A.M., 2001. Measuring usability with the use questionnaire12. Usability Interface 8 (2), 3–6.

Macindoe, J., Li, J.C., et al., 2014. Automatic circuit analysis problem and solution generation. 25th Annual Conference of the Australasian Association for Engineering Education: Engineering the Knowledge Economy: Collaboration, Engagement & Employability. School of Engineering & Advanced Technology, Massey University, p. 72.

Mellisy, D.A., Buechley, L., Resnick, M., Hartmann, B., 2016. Engaging amateurs in the design, fabrication, and assembly of electronic devices. DIS 2016 - Proceedings of the 2016 ACM Conference on Designing Interactive Systems: Fuse, pp. 1270–1281. https://doi.org/10.1145/2901790.2901833.

Miller, G.A., 1998. WordNet: An Electronic Lexical Database. MIT Press.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C.D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., Zeman, D., 2016. Universal Dependencies v1: a multilingual treebank collection. Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16). European Language Resources Association (ELRA), Portorož, Slovenia, pp. 1659–1666.

Qi, P., Dozat, T., Zhang, Y., Manning, C. D., 2019. Universal dependency parsing from scratch. arXiv preprint arXiv:1901.10457.

2019. Raspberry Pi.

Rabiner, L.R., Juang, B.H., 1986. An introduction to hidden Markov models. Vis. Ethnogr. 6 (2), 239–249. https://doi.org/10.12835/ve2017.2-0095.

2019. Scratch - Imagine, Program, Share.

Schlegel, V., Lang, B., Handschuh, S., 2019. Vajra : Step-by-step Programming with Natural Language, 30–39.

Schuster, M., Paliwal, K. K., 1997. Bidirectional recurrent neural networks as generative models 45 (May), 1–10. arXiv:1504.01575v2.

Shannon, C.E., 1948. A mathematical theory of communication. Bell Syst. Tech. J. 27 (April 1928), 379–423.623–656

Srinivasan, A., Dontcheva, M., Adar, E., Walker, S., 2019. Discovering natural language commands in multimodal interfaces, 661–672. 10.1145/3301275.3302292.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1929–1958.

Sung, E. J., Kim, K., Chung, H., You, J., 2019. Honk ? Talk !: designing driver-to-driver communication methods for social driving. Dis19, 295–299.

2019. Tinkercad | Create 3D digital designs with online CAD.

Thanisch, P., 1995. Natural language interfaces to databases - an introduction. Nat. Lang. Eng. 1 (1), 29–81. https://doi.org/10.1017/S135132490000005X.

Tomberlin, J.E., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. HLT-NAACL, pp. 252–259. https://doi.org/10.1007/BF02379273.

Toutanova, K., Klein, D., Manning, C.D., Singer, Y., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, pp. 252–259.

2019. VirtualBreadboard.

Waterhouse, C., 2016. Crossed wires: investigating the problems of end- user developers in a physical computing task. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16, 1, p. 38.

Williams, S., Power, R., Third, A., 2014. How easy is it to learn a controlled natural language for building a knowledge base? International Workshop on Controlled Natural Language. Springer, pp. 20–32.

Wu, T.-y., Yang, X.-d., 2019. Proxino : Enabling Prototyping of Virtual Circuits With Physical Proxies (c).

Yoshua Bengio, Patrice Simard, Paolo Frasconi, 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. 5 (2), 157.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., Toderici, G., 2015. Beyond short snippets: deep networks for video classification. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Zhan, Y., Hsiao, M., 2019. Multi-label classification on natural language sentences for video game design. 2019 IEEE International Conference on Humanized Computing and Communication (HCC). IEEE, pp. 52–59.

Zhu, K., Zhao, S., 2013. Autogami: a low-cost rapid prototyping toolkit for automated movable paper craft. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, pp. 661–670. https://doi.org/10.1145/2470654.2470748.