International Journal of Automation and Computing
16(4), August 2019, 462-474
DOI: 10.1007/s11633-018-1154-7

Synthesizing Robot Programs with Interactive Tutor Mode

Hao Li　　Yu-Ping Wang　　Tai-Jiang Mu

Tsinghua National Laboratory for Information Science and Technology (TNList), Department of
Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract: With the rapid development of the robotic industry, domestic robots have become increasingly popular. As domestic robots are expected to be personal assistants, it is important to develop a natural language-based human-robot interactive system for end-users who do not necessarily have much programming knowledge. To build such a system, we developed an interactive tutoring framework, named "Holert", which can translate task descriptions in natural language to machine-interpretable logical forms automatically. Compared to previous works, Holert allows users to teach the robot by further explaining their intentions in an interactive tutor mode. Furthermore, Holert introduces a semantic dependency model to enable the robot to "understand" similar task descriptions. We have deployed Holert on an open-source robot platform, Turtlebot 2. Experimental results show that the system accuracy could be significantly improved by 163.9% with the support of the tutor mode. This system is also efficient. Even the longest task session with 10 sentences can be handled within 0.7 s.

Keywords: Human-robot interaction, semantic parsing, program synthesis, intelligent robotic systems, natural language understanding.

1 Introduction

Modern domestic robots are expected to be personal assistants in the near future[1–3]. Although robots become more capable of performing different kinds of tasks[4, 5], it is still not convenient for end-users to instruct their robots to accomplish tasks as needed without programming knowledge.

Some studies help end-users with visual programming systems[6–8], which provide abstract building blocks embodying robot behavior. However, these systems still take the end-users a significant amount of time to learn the usage of each building block and to think about how to combine those blocks in a logical order.

Currently, building a natural language-based human-robot interactive system is increasingly appealing to end-users[9], because this is a rich and intuitive method with which end-users can offer sufficient and flexible instructions to support robot task planning[10, 11].

To develop such systems, many studies have focused on semantic parsing techniques that aim to translate natural language task descriptions into machine-interpretable logical forms. Traditional approaches[12–15] are often domain-specific or representation-specific. These approaches are heavily dependent on high-quality lexicons, manually-built templates and linguistic features[16]. Therefore, users of these systems are usually required to use a set of predefined keywords, templates and grammar. When it comes to undefined situations, the developer would have to manually add new keywords, templates or grammar to expand the representational ability of their system. Modern approaches[17–21] use machine learning methods to train a neural network for semantic parsing. However, these approaches rely on a large annotated dataset and efficient training algorithms during the training phase. Although the machine learning methods are general for different domains, they still need to collect and annotate domain-specific data and train new models.

Since the user demands change frequently, domestic robots are expected to perform different kinds of tasks as the user requires. However, previous semantic parsing methods rely on the developers to update their system for undefined situations. This process is time consuming and inconvenient for users, especially for emergency situations, because the developers must be involved to solve the problems. The ideal semantic parsing system should be capable of extending itself automatically to support users′ demands in undefined situations.

In this paper, we proposed an interactive tutoring framework, named "Holert", which allows users to *teach* the robot by further explaining their intentions. If the robot cannot understand a complex sentence in the task description, users can explain the complex sentence with several simple sentences. Holert can automatically identify the corresponding relationship between the explained sentences and the original one. This process runs recursively until the robot understands all of the sentences in a task description.

In order to realize this tutoring system, we defined a

Research Article
Manuscript received March 2, 2018; accepted August 8, 2018;
published online November 6, 2018
Recommended by Associate Editor James Whidborne
© Institute of Automation, Chinese Academy of Sciences and Springer-Verlag GmbH Germany, part of Springer Nature 2018

semantic dependency model to enhance the traditional templates. There are three advantages of introducing the dependency model. 1) The dependency model can help in finding the relationship between the user's original task description and the explanation dialogue, which is the key for the tutoring procedure. 2) After tutoring, the newly generated template enhanced by the dependency model can be applied not only to the demand just explained, but also to demands with similar grammatical structures. 3) Templates enhanced by the dependency model are more general, which could decrease the amount of manually designed initial templates.

In conclusion, the contribution of this paper lies in the following aspects:

1) We proposed an interactive tutoring approach for semantic parsing, which has two major advantages compared to previous approaches: a) Interactive tutor mode. In order to handle the user's demands in undefined situations, new templates can be generated from the guidance of users in an interactive procedure. b) Dependency model. The dependency model is carefully designed according to the characteristics of natural language. Templates enhanced by the dependency model can identify sentences that have the same semantic characteristics.

2) We implemented a framework named "Holert", which can synthesize robot programs from task descriptions in a short period of time. Furthermore, Holert works as a robot operating system (ROS)[22] node so that it can be easily deployed on different robot platforms.

3) We deployed Holert on an open-source robot platform, Turtlebot 2. We collected 129 unique navigation task descriptions from volunteers. The experimental results show that the system accuracy could be improved by 163.9% with the support of the tutor mode.

The rest of this paper is organized as follows. The motivation of this work is presented in Section 2. Section 3 explains the idea of Holert in detail. In Section 4, Holert was deployed on an open-source robot platform and evaluated with 129 task descriptions. Section 5 discusses the limitations and future works. Section 6 introduces the related work. Section 7 concludes this paper.

2 Motivation

Imagine a possible scenario for domestic robots where an aged man is confronted with a heart attack. He needs his robot to call the emergency center immediately, and reach out to neighbors for help.

In this situation, asking developers to implement a new robot application or using a visual programming system himself does not work, since it is time consuming and this aged man might not have enough time under heart attack conditions.

Natural language-based human-robot interactive systems would be a better way to complete the task. The aged man could describe his demand in natural language, such as "Call the emergency center. Go out of the house and ask the neighbors for help". Then the task description would be translated into robot instructions and executed exactly as desired.

With the traditional template-based approaches, the system may not understand every sentence. In this situation, the aged man has to ask the developers to update the system with new templates. This procedure may take days or even weeks to design and test the new templates, which is not acceptable in this case.

As to the neural network-based approaches, similar sentences may not exist in the training set. In this situation, the system may respond with incorrect actions, and the aged man would have to ask the developers to retrain the model by adding this case into the training set. This procedure may also take a lot of time.

In an ideal system, the aged man can further explain the original task "go out of the house" with simpler tasks, such as "go to the door of the house", "open the door" and "go 1 meter forward". If the system can understand these three simpler tasks, it should understand the original task. Furthermore, the system should also have the ability to understand similar sentences such as "go out of the room" without the need of further explanation next time.

To realize this system, some challenges remain to be solved:

1) In the tutoring procedure, how to find the corresponding relationship between the original sentence and the explained sentences automatically, and generate a new template for the original sentence?

2) How to design a general template that can cover as many similar sentences as possible?

3) The system should be highly accurate and efficient.

To overcome these challenges, we propose Holert. We will discuss the details of Holert in Section 3.

3 Approach

In this section, we will first introduce the architecture of Holert and then discuss the details of each part.

3.1 Architecture

As shown in Fig. 1, Holert works as a middle layer between users and robots. It is designed to accept task descriptions from users and output machine-interpretable logical forms to robots. Holert consists of four main parts: natural language (NL) parser, template library, tutor mode and synthesizer.

1) NL parser extracts dependency models from sentences of the task description. The dependency model is designed to express the inner dependency relations between word tokens of a sentence. This part will be presented in Section 3.2.

2) Template library maps dependency models to logic-

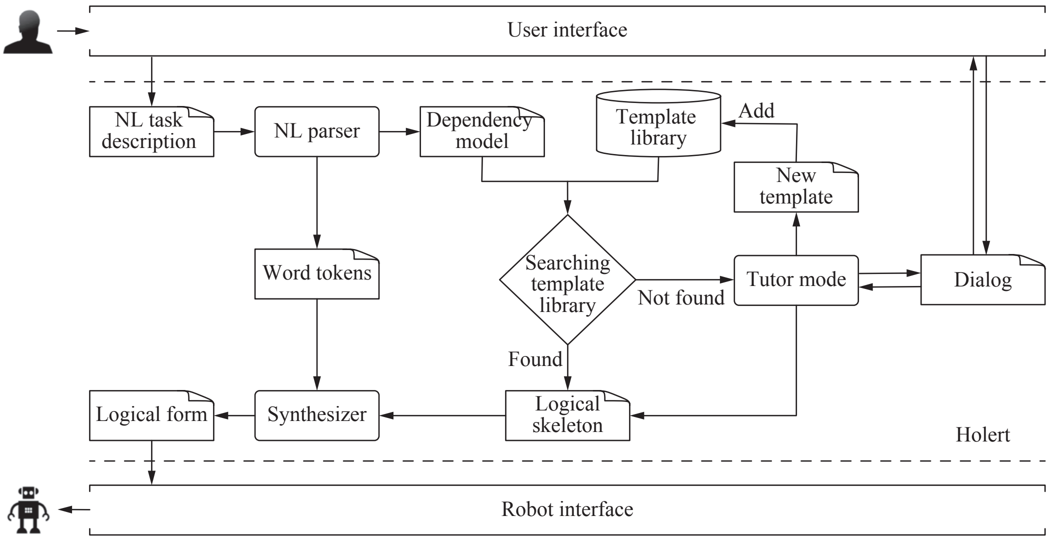


Fig. 1 Architecture of Holert. Holert accepts task descriptions from users and translates the natural language-based descriptions into machine-interpretable logical forms through four main parts: NL parser, template library, tutor mode and synthesizer.

al skeletons. A logical skeleton is the skeleton of a logical form. A key insight of Holert is that if two sentences share the same dependency model, they should have the same logical skeleton. This part will be presented in Section 3.3.

3) Tutor mode tries to learn new templates through dialogue with users. With the tutor mode, Holert can expand the template library automatically. This part will be presented in Section 3.4.

4) Synthesizer generates target logical forms from logical skeletons and word tokens of the original sentences. This part will be presented in Section 3.5.

Algorithm 1 shows the process of generating logical forms from task descriptions with Holert. First, a task description is split into a sequence of sentences $\boldsymbol{S}$ (Line 2). Second, the function *GetLogicSkeleton* is invoked to obtain a logical skeleton $\mathcal{LS}$ from each sentence $\mathcal{S}$ (Line 5). The function will first extract the dependency model of the sentence (Line 12) and then search the template library for a corresponding logical skeleton (Line 13). If no corresponding logical skeleton is found from the template library, tutor mode would be started to obtain the logical skeleton of this sentence (Line 15). Finally, the target logical form would be synthesized in Line 6 and added to the logical form set $\boldsymbol{LF}$ in Line 7.

Algorithm 1. Generating logical forms from task descriptions

Input: Original task description $\mathcal{T}$; template library $\boldsymbol{TL}$.
Output: Logic form set $\boldsymbol{LF}$ of original task description.

```
1) function MAIN($\mathcal{T}$)
2)   $\boldsymbol{S} \leftarrow SentencesAnnotation(\mathcal{T})$
3)   $\boldsymbol{LF} \leftarrow \varnothing$
4)   for each $\mathcal{S} \in \boldsymbol{S}$ do
5)     $\mathcal{LS} \leftarrow GetLogicSkeleton(\mathcal{S}, \boldsymbol{TL})$
6)     $\mathcal{LF} \leftarrow Synthesizer(\mathcal{S}, \mathcal{LS})$
7)     $\boldsymbol{LF} \leftarrow \boldsymbol{LF} + \mathcal{LF}$
8)   end for
9)   return $\boldsymbol{LF}$
10) end function
11) function GETLOGICSKELETON($\mathcal{S}$)
12)   $\mathcal{DM} \leftarrow GetDependencyModel(\mathcal{S})$
13)   $\mathcal{LS} \leftarrow \boldsymbol{TL}.get(\mathcal{DM})$
14)   if $\mathcal{LS} = \varnothing$ then
15)     $\mathcal{LS} \leftarrow TutorMode(\mathcal{S})$
16)   end if
17)   return $\mathcal{LS}$
18) end function
```

We will further explain the algorithm through the three task cases below. Suppose that Case 1 can find a matched template in the template library. Cases 2 and 3 are complex sentences for which Holert needs to learn new templates with the tutor mode.

Case 1. Move 5 meters forward.

Case 2. Give me the pen which you can pick up from the table.

Case 3. Give me the pen on the table.

3.2 NL parser and dependency model

We notice that even though some task descriptions are about different operations on different objects, such as "open the door" and "break the window", they share the same dependency relations between word tokens. So we can design a general model based on the dependency relations to cover similar task descriptions. This is the dependency model used in Holert. In this model, we replace word tokens with their part-of-speech (POS) tags[23], because POS tags are more general than word tokens. For example, verb tags can represent different operations like "open" and "break", while noun tags can represent different objects like "door" and "window".

The NL parser extracts the dependency model from a task sentence in four steps. Fig. 2 shows the process for Case 1.

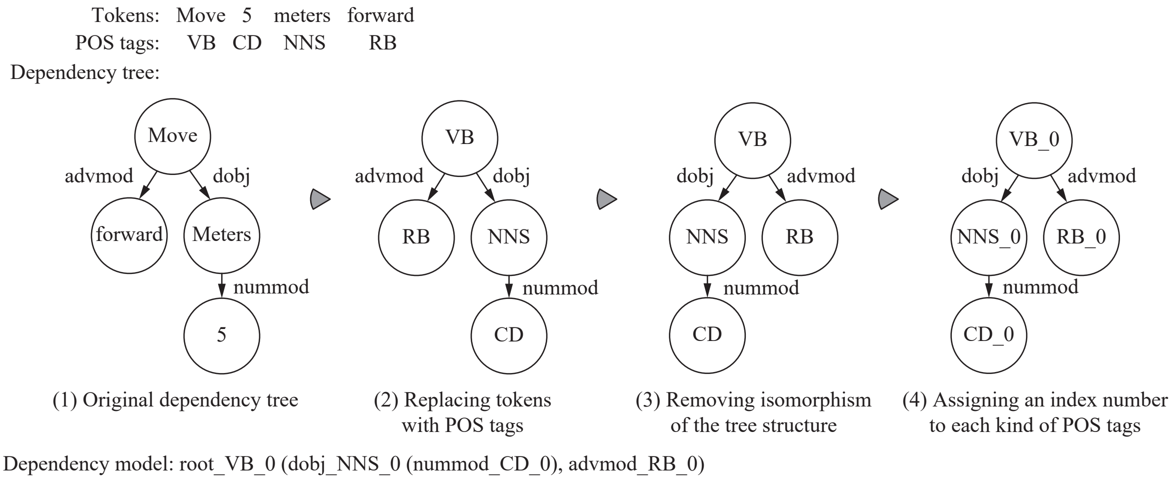

Tokens: Move 5 meters forward
POS tags: VB CD NNS RB
Dependency tree:
Move
advmod
dobj
forward
Meters
nummod
5
VB
advmod
dobj
RB
NNS
nummod
CD
VB
dobj
advmod
NNS
RB
nummod
CD
VB_0
dobj
advmod
NNS_0
RB_0
nummod
CD_0
(1) Original dependency tree
(2) Replacing tokens with POS tags
(3) Removing isomorphism of the tree structure
(4) Assigning an index number to each kind of POS tags
Dependency model: root_VB_0 (dobj_NNS_0 (nummod_CD_0), advmod_RB_0)

Fig. 2 Extracting the dependency model of the sentence in Case 1

1) We use Stanford natural language processing (NLP) tools[24] to split the sentence $\mathcal{S}$ into word tokens ($\boldsymbol{T} = T_1, T_2, \cdots, T_{|\mathcal{S}|}$), tag each token with POS tags ($\boldsymbol{P}_t = P_t^1, P_t^2, \cdots, P_t^{|\mathcal{S}|}$), and parse the sentence into a dependency tree. Each word token T_i corresponds to a node on the tree. The dependency tree provides dependency relations between word tokens in the sentence. The dependency relation is defined as a triplet $\langle T_i, D_i^j, T_j \rangle$, where D_i^j is the dependency relation between the governor node T_j and the dependent node T_i. For example, the dependency relation between nodes "Move" and "forward" is represented as $\langle \text{forward}, \text{advmod}, \text{Move} \rangle$.

2) We replace word tokens with their POS tags. Therefore, the dependency relation triplet is transferred to $\langle P_t^i, D_i^j, P_t^j \rangle$. For example, the token "Move" is replaced by "VB", which means a verb, and "forward" is replaced by "RB", which means an adverb.

3) In order to remove the isomorphism of the tree structure, we give each node a hash value and keep an increasing order between sibling nodes.

4) We traverse the tree in order and assign an index number I_t^i to each kind of POS tag according to the access sequence, so that each node on the tree is unique. That means the first visited "NN" node is assigned an index number "0", and the second visited "NN" node is assigned an index number "1", etc. We define $P_t^i\_I_t^i$ as the marker M_i of the word token T_i. After this step, the tree is called a dependency model in this paper.

To find a dependency model more quickly, we use an equivalent string-representation of the dependency model instead of the tree-representation. The string-representation is an in-order traversal of the tree. We first visit the root of the tree and then visit its children recursively from left to right. $D_i^j\_M_i$ are used to represent the visited nodes T_i, where D_i^j is the dependency relation between T_i and its governor node T_j, and M_i is the marker of T_i. Specifically, the dependency relation of the root node is represented as "root". Parentheses are used to show the governor-dependent relations between nodes.

For example, the string-representation of the dependency model for "Move 5 meters forward" is:

root_VB_0 (dobj_NNS_0 (nummod_CD_0), advmod_RB_0).

3.3 Template library

The template library is a set of templates. Each template is a key-value pair, where the key is a dependency model and the value is a corresponding logical skeleton.

The template library is initialized manually based on the instructions of the target robot. We construct the initial template library with three steps:

1) List all of the instructions of the target robot and collect different expressions of the instruction in natural language descriptions.

2) Extract dependency models from these descriptions.

3) Assign each dependency mode with a corresponding logical skeleton manually.

For example, suppose the target robot has a "Move" instruction, and the instruction has three parameters, including an integer type distance value, a measure unit and a direction. The possible natural language expression of the instruction could be "go 35 feet backward". For this task description, the robot is expected to execute the instruction "Move(35, feet, backward)". We first extract the dependency model of the task description. Then we replace the name and parameters of this instruction with corresponding markers in the dependency model. For example, "go" is the action word that indicates the "Move" instruction and its marker is "VB_0", so we replace "Move" with "VB_0". Similarly, we replace the parameters with corresponding markers "CD_0", "NNS_0" and "RB_0". Finally, we manually construct a template for "go 35 feet backward", which is shown in Fig. 3.

3.4 Tutor mode

In this subsection, we will explain the tutor mode using Cases 2 and 3.

For Case 2, “Give me the pen which you can pick up from the table”, Holert would start a dialogue with the user (see Fig. 4). Holert asked the user to further explain the task. The user then split the task into two smaller tasks: “Pick up the pen on the table” and “Give me the pen”. Then, Holert would generate a new template for the original task with the supplementary information and add the new template into the initial library. When the user asks the robot to “Give me the scarf which you can pick up from the hanger” the next time, Holert could find the learned template from the library directly.

Algorithm 2. Generate a new template in tutor mode

Input: Original task sentence $\mathcal{S}$, dependency model of the sentence $\mathcal{DM}_{\mathcal{S}}$, template library $\boldsymbol{TL}$.

Output: The new template $\mathcal{NT}$ of the original task sentence.

```
1) $\boldsymbol{T} \leftarrow GetExplanationFromUser(\mathcal{S})$
2) $\boldsymbol{S} \leftarrow SentencesAnnotation(\boldsymbol{T})$
3) $\boldsymbol{LS} \leftarrow \varnothing$
4) for each $\widetilde{S} \in \boldsymbol{S}$ do
5)     $\widetilde{LS} \leftarrow GetLogicSkeleton(\widetilde{S})$
6)     $\overline{LS} \leftarrow ReplaceMarker(\widetilde{LS})$
7)     $\boldsymbol{LS} \leftarrow \boldsymbol{LS} + \overline{LS}$
8) end for
9) $\mathcal{NT} \leftarrow GenerateNewTemplate(\mathcal{DM}_{\mathcal{S}}, \boldsymbol{LS})$
10) $\boldsymbol{TL} \leftarrow \boldsymbol{TL} + \mathcal{NT}$
```

Algorithm 2 shows how to generate a new template in tutor mode. Fig. 5 shows the process of generating a new template for Case 2 with this algorithm. There are four steps:

1) Ask the user for further explanation $\boldsymbol{T}$ of the original task sentence (Line 1). Split the explanation into a sequence of sentences $\boldsymbol{S}$ (Line 2).

```
root_VB_0 (
    dobj_NNS_0 (
        nummod_CD_0
    ),
    advmod_RB_0
)
```

→

```
{
    call VB_0 [
        CD_0 ,
        NNS_0 ,
        RB_0
    ]
}
```

Key : dependency model

Value : logic skeleton

Fig. 3 A template example

User: Give me the pen which you can pick up from the table.

Robot: Sorry, can you explain the order?

User: Pick up the pen from the table. Give me the pen.

Robot: OK, I see.

– – – – – – – – – –A few days later– – – – – – – – – – –

User: Give me the scarf which you can pick up from the hanger.

Robot: OK.

Fig. 4 A dialogue example with the user

2) For each sentence in $\boldsymbol{S}$, invoke the function *GetLogicSkeleton* of Algorithm 1 to generate the corresponding logical skeleton $\widetilde{LS}$ (Line 5). If some explanation sentences are still too complex, the tutor process will run recursively until Holert “understands” all of the explanation sentences. For Case 2, the explanation contains two sentences, and the corresponding logical skeletons 1 and 2 (see Fig. 5) would be found in the template library.

3) Replace the markers in $\widetilde{LS}$ with corresponding markers of the original sentence (Line 6). Suppose $P_t^1\_I_t^1, P_t^2\_I^2, \cdots, P_t^n\_I_t^n$ are the markers in $\widetilde{LS}$, and $T_1, T_2, \cdots, T_n$ are the tokens in the explanation sentences for those markers. Holert would find the same token (see the dotted line in Fig. 5) appearing in the original sentence and replace $P_t^1\_I_t^1, P_t^2\_I^2, \cdots, P_t^n\_I_t^n$ with the corresponding markers for these tokens in the original sentence. For example, there are three markers in the logical skeleton 1: “VB_0”, ”NN_0” and “NN_1”. The tokens for the three markers are “pick”, “pen” and “table”. We searched the original sentences for the three tokens and found that their corresponding markers were “VB_1”, “NN_0” and “NN_1”, respectively. Then, we replaced “VB_0” in logical skeleton 1 with “VB_1”.

4) Generate a new template, where the key is the dependency model of the original sentence and the value is the combination of the marker-replaced logical skeletons in a sequence order (Line 9). Finally, add the new template into the template library (Line 10).

There are two special situations when using the tutor mode. One is that the explanation from the user contains only a single sentence. The other is that action verbs are missing in the original sentence.

Explanations in a single sentence. The user may explain the original sentence with only one sentence. For example, the user may explain “can you open the door?” with “open the door for me”, and Holert cannot find a template for “open the door for me” from the library either. A radical strategy would be used to find a similar template. The strategy is based on the experience that some tokens in the sentence perform dependency roles but contribute nothing to the semantic meaning. For example, we can safely delete the two tokens “for” and “me”, because they do not affect the meaning of the sentence. When using the radical strategy, Holert would enumerate all of the cases of deleting some tokens (from 1 token to half of all tokens) in the sentence and search for the dependency models of those sentences in the template library. If matched, the sentence becomes a candidate to be presented to the user. The user then chooses one of them to continue the tutor process above. If none of them meet the user requirement, the tutor process has failed. Holert would record this sentence for manually constructing a new template by developers.

Missing actions. Some complex task descriptions contain missing actions, such as Case 3. “Give me the pen on the table” is another way of expressing the task in

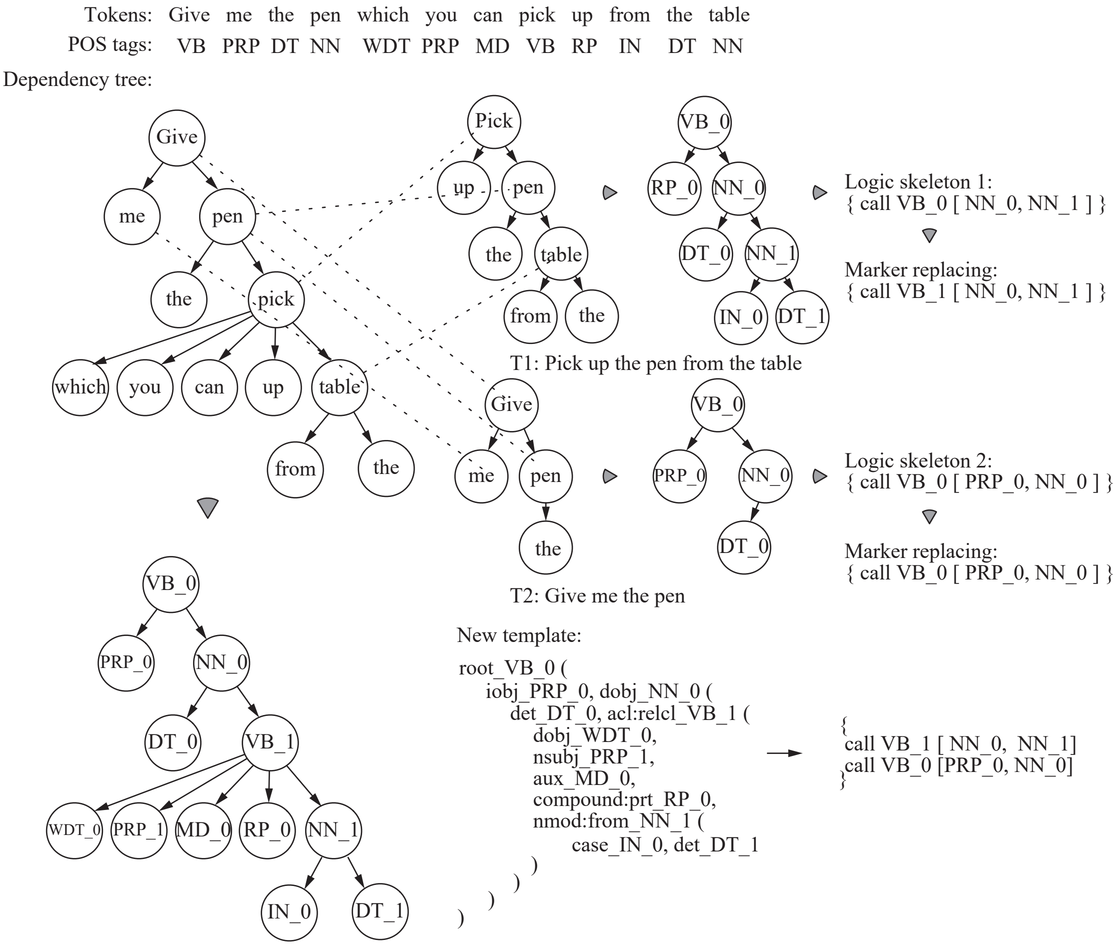


Fig. 5 Generating a new template for Case 2

Case 2. In order to give the pen to the user, the robot should pick up the pen from the table first. If the user still splits the task into the same two smaller tasks (see Fig. 6), then Holert cannot find a corresponding token "pick" in the original sentence when generating a new template. We use "*Unknown*" in the generated new template to represent a missing action. If the user asks the robot to "Give me the scarf on the hanger" the next time, Holert would find the matched template in the library and try to infer the missing action by program synthesis techniques in Section 3.5.

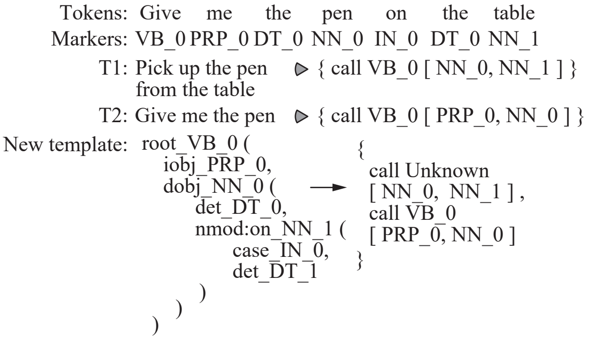

Tokens: Give me the pen on the table
Markers: VB_0 PRP_0 DT_0 NN_0 IN_0 DT_0 NN_1
T1: Pick up the pen from the table
{ call VB_0 [NN_0, NN_1] }
T2: Give me the pen
{ call VB_0 [PRP_0, NN_0] }
New template: root_VB_0 (
iobj_PRP_0,
dobj_NN_0 (
det_DT_0,
nmod:on_NN_1 (
case_IN_0,
det_DT_1
)
)
)
{
call Unknown
[NN_0, NN_1],
call VB_0
[PRP_0, NN_0]
}

Fig. 6 Generate a new template for Case 3, "Unknown" is used to represent the missing action

3.5 Synthesizer

The synthesizer tries to fill the holes in the logical skeleton with information from the task description. There are two kinds of holes in a logical skeleton, 1) parameter marker and 2) function marker. The synthesizer first assigns the parameter markers with typed values extracted from the task description and then finds the best-matched robot instruction to instantiate the function marker.

Parameter marker assignment. The parameter markers contain node information from which we can find the corresponding tokens in the original task descriptions. The synthesizer first uses the general regular expressions to identify the typed values from the tokens. If that process fails, then it would search in a static mapping table. If both have failed, the synthesizer would record the sentence for manually constructing a new template by developers and report with an error message. The details of the two identification methods are as follows:

1) Regular expressions. Some types of data can be easily recognized using regular expressions, such as integers, dates, phone numbers, etc.

2) A static mapping table. Such as "last one", which would be mapped to an integer type value "–1", and "meter", which would be mapped to a measurement type value.

Table 1 shows all of the parameters found in the three cases, such as the token "5" for marker "CD_0", which is mapped to an integer type value based on regular expressions, and the token "meters" for marker "NNS_0", which is mapped to a measurement type value based on the static mapping table.

Function marker instantiation. After assigning all parameter markers with typed values, the synthesizer tries to find a robot instruction for the function marker. In order to find the correct robot instruction to instantiate the function marker, we need to annotate the robot instructions with two kinds of information:

1) Parameter type signatures of the instruction. This gives the information that determines how many parameters this instruction would take in and what their types are.

2) Keywords to summarize the function of the instruction. We will extend the keywords set automatically by searching their synonyms from WordNet[1].

The instantiation of the function markers is robot-dependent. Suppose the target robot has 5 instructions as shown in Table 2. The synthesizer would instantiate the function markers in two steps. First, the synthesizer finds a set of instruction candidates that are possible to instantiate the function marker. Then, it selects the best one by a ranking strategy.

Algorithm 3. Choosing the best robot instruction

Input: Parameter type set $\boldsymbol{P}$; robot instruction set $\boldsymbol{I}$.

Output: The best-matched robot instruction $\mathcal{I}$ to replace the function marker.

```
1) $\boldsymbol{C} \leftarrow \varnothing$                /*Candidate set*/
2) for each $\mathcal{I} \in \boldsymbol{I}$ do
3)   $\boldsymbol{P}_{\mathcal{I}} \leftarrow GetParameterTypeList(\mathcal{I})$
4)   if $|\boldsymbol{P}_{\mathcal{I}}| = |\boldsymbol{P}|$ then
5)     $\widetilde{P}_{\mathcal{I}} \leftarrow \{\mathcal{P}^i_{\mathcal{I}} | \mathcal{P}^i_{\mathcal{I}} \in \boldsymbol{P}_{\mathcal{I}}, \mathcal{P}^i \in \boldsymbol{P}, \mathrm{type}(\mathcal{P}^i_{\mathcal{I}}) \neq \mathrm{type}(\mathcal{P}^i)\}$
6)     if $\widetilde{P}_{\mathcal{I}} = \varnothing$ then
7)       $\boldsymbol{C} \leftarrow \boldsymbol{C} + \mathcal{I}$
8)     end if
9)   end if
10) end for
11) if $\boldsymbol{C} = \varnothing$ then
12)   ReportError()
13)   return $\varnothing$
14) end if
15) $\widetilde{\boldsymbol{C}} \leftarrow SortByRanking(\boldsymbol{C})$ /*Ranking strategy based on tokens*/
16) return $\widetilde{\boldsymbol{C}}[0]$
```

Algorithm 3 shows the process. The candidates $\boldsymbol{C}$ should satisfy the type constraint, which means: a robot instruction is a potential choice for instantiating the function marker only when 1) the instruction has the same number of parameters as the number of parameter markers in the logical skeleton (Line 4), and 2) there are values that are identified and whose types are the same as the type signatures of the instruction parameters (Lines 5–8).

We use a ranking strategy to choose the best-matched one from $\boldsymbol{C}$ (Line 15). We first collected all corresponding tokens for the markers in the logical skeleton, which includes a list of parameter marker tokens and a function marker token. If the function marker is "*Unknown*", it

Table 1 Parameters found in the three cases

Exclude	Marker	Token	Value	Type	Method
1	CD_0	5	5	Integer	Regular expression
1	NNS_0	meters	meter	Measurement	Static mapping table
1	RB_0	forward	forward	Direction	Static mapping table
2, 3	PRP_0	me	Master	Person	Static mapping table
2, 3	NN_0	pen	pen	SmallObject	Static mapping table
2, 3	NN_1	table	(x, y)	Coordinate	Static mapping table

Table 2 Instructions provided by the target robot

Number	Instruction	Type signatures	Keywords
1	Move	Integer, Measurement, Direction	MOVE, GO, WALK
2	GrabFromTable	SmallObject, Coordinate	GRUB, PICK, TABLE
3	GrabFromHanger	SmallObject, Coordinate	GRUB, PICK, HANGER
4	Drop	SmallObject, Coordinate	DROP, LAY, OFF
5	Give	SmallObject, Person	GIVE, BRING

---

[1] WordNet provides a synonym searching service for English. Refer to https://wordnet.princeton.edu/ for further information.

would be omitted. Then, the synthesizer would rank the instruction candidates by comparing the collected tokens with the keywords of each instruction candidate. A candidate will get a higher score if its keyword tags match more word tokens. Finally, the synthesizer would choose the best-matched instruction with the highest score.

Table 3 shows the synthesized logical forms for the three cases. For example, the logical skeleton "call VB_1 [NN_0, NN_1]" of Case 2 has two parameter markers, "NN_0" and "NN_1". Their corresponding tokens are "pen" and "table", which would be mapped to a SmallObject type value and a Coordinate type value (see Table 1). According to Algorithm 3, there are three candidates to instantiate "VB_1", i.e., instructions 2–4 in Table 2, because they all satisfy the type constraint. The *GrabFromTable* instruction gets the highest score with the ranking strategy, since it has two matched keywords, "PICK" and "TABLE". As a result, the function marker "VB_1" is instantiated with *GrabFromTable*. Finally, we get the synthesized logical form "call GrabFromTable [pen, (x, y)]".

4 Evaluation

4.1 Environment setup

Preparing on robot. We deploy Holert on a Turtlebot 2 robot. This robot has a Kobuki base, which supports basic movement instructions. We use ROS as the message-passing system on this robot. Holert works as a ROS node that receives user task descriptions and publishes robot instructions. We also implemented two assistant nodes to make the system more practical. One is the user-interface node that interacts with users by automatic speech recognition (ASR) and text-to-speech (TTS) tools. The other is the robot-control node that translates the logical form to robot instructions and maintains a navigation instruction queue to be executed.

The Kobuki basic movement instructions are machine-related and control the linear and angular velocity directly. In order to enable the users to interact naturally with the system, we defined and implemented 8 user-friendly instructions (see Table 4). Those instructions are close to natural language.

Constructing the initial template library. We tagged the 8 extended instructions with type signatures and keywords as shown in Table 4. For each instruction, we use multiple natural language task descriptions to express it. For example, the *rotate* (*Direction*) instruction can be expressed as "turn left", "turn to the left" or "make a left turn". Then we extract the dependency models of these descriptions and assigned a logical skeleton for each manually, which make up the initial template library.

Table 3 Logic forms for the three cases

Case	Logic skeleton	Logic form
1	{call VB_0 [CD_0, NNS_0, RB_0]}	{call Move [5, meter, forward]}
2	{	{
	call VB_1 [NN_0, NN_1],	call GrabFromTable [pen, (x, y)],
	call VB_0 [PRP_0, NN_0]	call Give [pen, Master]
	}	}
3	{	{
	call Unknown [NN_0, NN_1] ,	call GrabFromTable [pen, (x, y)],
	call VB_0 [PRP_0, NN_0]	call Give [pen, Master]
	}	}

Table 4 Extended movement instructions for Turtlebot 2

Number	Instruction	Brief descriptions	Parameter type signatures	Keywords
1	go	Keep going until receiving the stop instruction	Direction	GO, WALK, MOVE
2	go	Walk a certain distance	Integer, Measurement	GO, WALK, MOVE
3	rotate	Turn left/right/back	Direction	TURN, ROTATE
4	rotate	Turn left/right at a certain degree/radian	Direction, Integer, Measurement	TURN, ROTATE
5	speed	Speed up/down	Boolean	SPEED, FAST, QUICK, SLOW
6	rotateSpeed	Rotate speed up/down	Boolean	ROTATE, TURN, SPEED, FAST, SLOW
7	charge	Go to charge itself	Coordinate	CHARGE
8	stop	Stop action	$\varnothing$	STOP, QUIT, SHUTOFF

Collecting task descriptions. We invited 10 volunteers to interact with the robot and to give 20 navigation instructions. The volunteers were separated and did not know the dialogue content of others. Finally, we obtained 129 unique navigation instructions.

4.2 Accuracy

Table 5 shows the experimental accuracy of Holert on the 129 user-collected cases. In order to test the Holert node individually, we manually omit the influence of speech recognition errors. In practice, all the dialogues (text mode) are delivered to a supervisor (human) at the same time. If the speech recognition errors appeared from the user-interface node, the supervisor would ask the users to repeat their requests. We first run Holert on the 129 cases without the tutor mode. Only 36 out of 129 (27.9%) can be handled directly using the initial template library. Then, we turn on the tutor mode partly without the support of the two special situations that were discussed in Section 3.4. Holert can successfully synthesize correct robot instructions for 79 (61.2%) cases. Finally, we turn on the support for the two special situations, and the accuracy rate has been further increased to 73.6% (95 out of 129). Overall, the experimental result shows that the system accuracy could be improved by 163.9% with the support of tutor mode.

Table 5 Accuracy of the 129 user-collected cases

	Without tutor mode	Tutor mode (partly)	Tutor mode
Successful cases	36	79	95
Accuracy	27.9%	61.2%	73.6%

4.3 Efficiency

In Holert, a task description would be split into a set of sentences, as the sentences are the basic units to be handled. As shown in Fig. 7, the more sentences included in a task session, the more execution time is needed, and this relation is almost linear. Besides, Holert can respond to users in a short period of time. Most of the task sessions, which contain no more than 4 sentences, could be handled within 0.2 s. Even the longest task session with 10 sentences could be handled within 0.7 s.

4.4 User effort

We first define the user effort of explaining a single sentence and a task. Then we will show the user effort statistics of explaining the 129 user-collected task descriptions.

In Holert, the user effort of explaining a single sentence $\mathcal{S}$ is defined as the depth of the dialogue $D(\mathcal{S})$ (Q/A rounds to understand the sentence), which is calculated using (1). The user effort of explaining a task is equal to explaining its most complicated sentence.

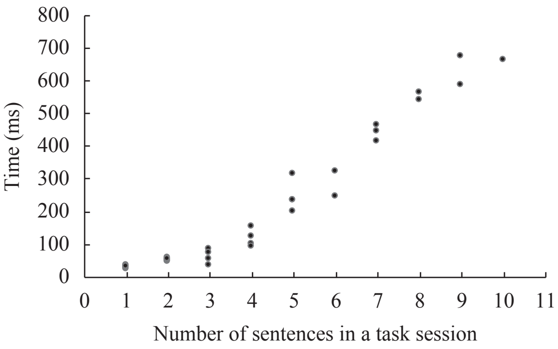


Fig. 7 Efficiency analysis. Each black dot in the figure represents a task. The X-axis is the number of total sentences handled in the task session, which includes sentences both in the original task description and in the tutor mode dialogue. The Y-axis is the execution time in milliseconds. The user's response time in tutor mode is omitted.

$$D(\mathcal{S}) = \begin{cases} 0, \\ \quad \text{if the dependency model of } \mathcal{S} \text{ falls in the} \\ \quad \text{template library} \\ 1, \\ \quad \text{if explained in a single sentence} \\ \text{Max}(D(\mathcal{S}_1), D(\mathcal{S}_2), \cdots, D(\mathcal{S}_n)) + 1, \\ \quad \text{if explained in multiple sentences } \mathcal{S}_1\mathcal{S}_2\cdots\mathcal{S}_n. \end{cases}$$

As shown in Table 6, Holert is easy to use. In most situations (117/129), the tutor process can end up with no more than 2 Q/A rounds, and the robot can successfully understand most of them (92/117). In a few cases (12/129), when the user effort reaches 3 (4), it becomes more and more difficult to find a corresponding mapping relation between the original sentence and the explained sentences. As a result, the accuracy of Holert rapidly dropped to 30% (0%).

Table 6 User effort of explaining the 129 tasks

User effort (Depth of dialogue)	Count	Successful cases	Accuracy
0	36	36	100%
1	55	40	72.7%
2	26	16	61.5%
3	10	3	30%
4	2	0	0%
Total	129	95	73.6%

4.5 Successful task cases

4.5.1 Simple tasks without tutor mode

Table 7 shows some simple task cases that could be handled directly without tutor mode. The second column gives the task description. The third column lists the instructions involved. There are 43 templates in the initial

Table 7 Simple task cases without tutor mode

No.	Task description	Instruction	Template ID
1	Move 3 meters ahead	go	3
2	Turn back	rotate	12
3	Turn 45 degrees to the left	rotate	17
4	Go 5 meters forward. Turn back.	go, rotate	3, 12
5	Turn left at 45 degrees. Move.	rotate, go	19, 1
	Move slower. Stop. Go to charge.	speed, stop, charge	24, 1, 41

template library. The last column shows the matched template IDs for these task sentences.

As shown in the last column of Table 7, the dependency model is capable of expressing multiple task types. For example, “Move” and “Stop” share the same matched template. “Move 3 meters ahead” and “Go 5 meters forward” also share the same matched template.

4.5.2 Complex tasks with tutor mode

We choose three complex task cases to show the tutor mode in different situations.

Explanations in a single sentence. As shown in Fig. 8, the user ordered the robot to turn back with “Can you turn back” and explained it in only one sentence “Turn back please”. However, the initial library does not contain the matched template. Therefore, Holert starts the radical strategy to randomly delete some nodes in the dependency tree of the sentence and get a list of potential candidates ([“turn back”, “turn please”, “back please”]). After searching each candidate in the template library, only “turn back” has a matched template. So Holert asked the user for confirmation. Then, it used “turn back” as the explanation for “Can you turn back”. Finally, the tutor mode generated a new template for “Can you turn back” and executed the order successfully.

User:	Can you turn back?
Robot:	Sorry, can you explain the order?
User:	Turn back please.
Robot:	Do you want me to turn back?
User:	Yes.
Robot:	OK, I see.

Fig. 8 Part of the dialogue with the user (I)

Conjunction words. Fig. 9 shows how to handle the conjunction words, such as “and”, in Holert. The user ordered the robot to “Turn 30 degrees left and move ahead”. Although it is straightforward for humans that the task can be split into two smaller tasks of “turn 30 degrees left” and “move ahead”, Holert cannot understand the word “and”. Since Holert neither analyzes the sentence grammatically nor tries to understand the dependency relations semantically, it only extracts the dependency model for the entire sentence and cannot find a matched template. In this case, the tutor mode is needed.

User:	Turn 30 degrees left and move ahead.
Robot:	Sorry, can you explain the order?
User:	Turn 30 degrees left. Move ahead.
Robot:	OK, I see.

Fig. 9 Part of the dialogue with the user (II)

User:	Turn back after going 1 meter to the left.
Robot:	Sorry, can you explain the order?
User:	Go 1 meter to the left. Turn back.
Robot:	Sorry, can you explain the order "Go 1 meter to the left"?
User:	Turn left. Go 1 meter forward.
Robot:	OK, I see.

Fig. 10 Part of the dialogue with the user (III)

Explain in split sentences recursively. As shown in Fig. 10, the user gave a complex task sentence “Turn back after going 1 meter to the left”. Holert searched the template library and failed to find a matched template. Therefore, tutor mode is started. The user explained the sentence with two simpler sentences. 1) Go 1 meter to the left. 2) Turn back. However, Holert cannot find a matched template for the first sentence. So it starts the tutor process for the first sentence recursively. After learning the template of the first task, Holert generates the final template for the original task sentence.

4.6 Analysis on failed cases

We analyzed the failed cases and found three main reasons causing these failures:

1) Undefined robot instructions. For example, the user asked the robot to “go upstairs”. However, the robot can only walk on the ground and does not provide an instruction like “upstairs”. In this case, Holert could only find the corresponding logical skeleton but failed to synthesize the correct logical form.

2) Undefined template for robot instructions. The template library is the knowledge base of Holert. However, the initial temporary library is constructed manually. Although efforts have been made to express the 8 basic instructions with descriptions that have different dependency models, it is still highly possible that we would miss some of them, since the natural language is too flexible to enumerate them all.

3) Tiny changes to the dependency model. Holert is dependency-sensitive. The dependency model is extracted based on the Stanford dependency analysis of the task sentences. The Stanford parser defined 45 main dependency relations between the governor and dependent. Tiny changes to the sentence could lead to a different dependency model. For example, “turn left” and “turn left please” have the same semantic meaning but their dependency models are different.

5 Limitations and future works

In addition to the limitations discussed in Section 4.6, Holert has another three limitations.

Although the enhanced template used in Holert is domain-independent, we still need to manually annotate the robot instructions with parameter type signatures and some keywords to support the synthesis algorithm. In the future, we will try to extract the parameter type information and keywords automatically from the robot instruction documents.

Another limitation results from the ambiguity of natural language. The extraction of the dependency model depends on the correct dependency analysis by NLP tools. Although Holert uses the state-of-art Stanford NLP tools, the parser may fail to extract the correct dependency model because of the ambiguity of natural language. If the parser made the wrong analysis, Holert would get the wrong result as well. We will pay attention to the development of NLP techniques and keep updating Holert with better NLP tools.

Finally, a radical strategy is used in the situation of explaining in a single sentence, which aims to overcome the dependency-sensitive problem. The strategy would randomly delete some nodes from the dependency tree, with the hope that the remaining tree not only reserves the semantic meaning of the original sentence but also has a corresponding template in our library. However, the number of nodes to be deleted is at most half of the length of the sentence, which is an empirical value. If we delete too many nodes, there is a risk that the semantic meaning of the original sentence would be changed. Therefore, this solution is not good enough. In our future work, we will try to find a better strategy to solve the dependency-sensitivity problem, such as slightly modifying the dependency tree of the original sentence to find a "nearby" logical skeleton in the template library.

6 Related work

Human-robot interactive systems are a critical and widely-studied aspect of domestic robots, and natural language instructions are a key component of human-robot interaction. Previous studies have treated the task as a problem of parsing natural language descriptions into machine-interpretable logical forms.

Several systems parse natural language descriptions to sequences of atomic actions that must be grounded into fully specified world models. Look et al.[25] presented an ontology that addressed the problem of expressing the semantics that define how a particular space is used. Kollar et al.[26] extracted a sequence of spatial description clauses from natural language input with given information about the environmental geometry and detected visible objects, and then used a probabilistic model to connect them together to find a path for the robot. These approaches assume an initial knowledge map, which describes a complex indoor environment with object and land-marks. Other systems are heavily dependent on high-quality lexicons, manually-built templates or linguistic features. For example, Bugmann et al.[27] implemented an instruction-based learning (IBL) system with 15 primitive functions, each of which had a fixed parameter list. MacMahon et al.[28] inferred a set of pre-defined actions from the knowledge of both linguistic conditional phrases and local configurations. Rybski et al.[29] proposed a system that can learn simple action scripts from natural language, but the instructions must follow a pre-defined grammar. Matuszek's work[11] trained a parser based on example pairs of natural language orders and corresponding control language expressions. The parser allowed a robot to interpret navigation instructions online while moving through a previously unknown environment. In contrast to our work, users of these systems are usually required to use a set of predefined keywords, templates and grammar. Besides, most of these approaches rely on a manually constructed parser, rather than learning relations from dialogues.

Some systems also teach robots new knowledge through dialogues. They are similar to how our tutor mode works but with a different focus. Thomason et al.[30] used a three-component (action, patient and recipient) template to support two kinds of robot actions (walking and bringing items). The learning system in this work focused on lexicons. It learned new lexical items by mapping them to the three components through dialogues. She et al.[31] focused on learning new actions. Through dialogues, users of this system can teach the robot how to construct complicated actions with three primitive actions. Unlike this, Holert focuses on the grammatical structure of natural language. It learns new grammatical structures through finding the relationship between the dependency models of original sentences and explanation dialogues.

Recently, some researchers have tried to solve the semantic parsing problem with machine learning techniques. Dong and Lapata[16] presented an encoder-decoder neural network model for mapping natural language descriptions to their meaning representations. Their model encoded natural language descriptions into vectors and generated logical forms as sequences or trees using recurrent neural networks with long short-term memory units. Grefenstette et al.[32] proposed a different architecture for semantic parsing based on the combination of two neural network models. The first model learned the shared representations from pairs of questions and their translations into knowledge-based queries, whereas the second model generated the queries conditioned on the learned representations. However, these approaches rely on big annotated datasets and efficient training algorithms in the training phase. Besides, adversarial inputs would make them vulnerable and cause the neural networks to misclassify[33, 34].

7 Conclusions

In this paper, we proposed an interactive tutoring framework named "Holert" for human-robot interaction. The main idea is to enable the robot to learn from end-user explanations under a novel designed tutor mode. With the tutor mode, the users can teach their robot new skills through dialogues. Developer supports are not needed in this process, which would save a significant amount of time and manual work. The experimental result on Turtlebot 2 shows that Holert can successfully synthesize correct instructions for 73.6% of the task descriptions collected and that the system accuracy could be improved by 163.9% with the support of the tutor mode. Furthermore, the process is efficient. Even the longest task session, which contains 10 sentences, can be handled within 0.7 s.

Acknowledgements

This work was supported by Tsinghua University Initiative Scientific Research Program (No. 20141081140).

References

[1] J. Scholtz. Theory and evaluation of human robot interactions. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, IEEE, Big Island, USA, 2003. DOI: 10.1109/HICSS.2003.1174284.

[2] I. G. Alonso, M. Fernández, J. M. Maestre, M. del Pilar Almudena García Fuente. *Service Robotics within the Digital Home: Applications and Future Prospects*, Dordrecht, Holland: Springer, 2011. DOI: 10.1007/978-94-007-1491-5.

[3] R Borja, J. R. De La Pinta, A. Álvarez, J. M. Maestre. Integration of service robots in the smart home by means of UPnP: A surveillance robot case study. *Robotics and Autonomous Systems*, vol. 61, no. 2, pp. 153–160, 2013. DOI: 10.1016/j.robot.2012.10.005.

[4] C. Zhou, M. H. Jin, Y. C. Liu, Z. Zhang, Y. Liu, H. Liu. Singularity robust path planning for real time base attitude adjustment of free-floating space robot. *International Journal of Automation and Computing*, vol. 14, no. 2, pp. 169–178, 2017. DOI: 10.1007/s11633-017-1055-1.

[5] K. C. D. Fu, Y. Nakamura, T. Yamamoto, H. Ishiguro. Analysis of motor synergies utilization for optimal movement generation for a human-like robotic arm. *International Journal of Automation and Computing*, vol. 10, no. 6, pp. 515–524, 2013. DOI: 10.1007/s11633-013-0749-2.

[6] S. Alexandrova, Z. Tatlock, M. Cakmak. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *Proceedings of IEEE International Conference on Robotics and Automation*, Seattle, USA, pp. 5537–5544, 2015. DOI: 10.1109/ICRA.2015.7139973.

[7] C. Datta, C. Jayawardena, I. H. Kuo, B. A. MacDonald. RoboStudio: A visual programming environment for rapid authoring and customization of complex services on a personal service robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, pp. 2352–2357, 2012. DOI: 10.1109/IROS.2012.6386105.

[8] J. Li, A. Q. Xu, G. Dudek. Graphical state space programming: A visual programming paradigm for robot task specification. In *Proceedings of IEEE International Conference on Robotics and Automation*, Shanghai, China, pp. 4846–4853, 2011. DOI: 10.1109/ICRA.2011.5979630.

[9] M. A. Goodrich, A. C. Schultz. Human- robot interaction: A survey. *Foundations and Trends in Human-computer Interaction*, vol. 1, no. 3, pp. 203–275, 2007. DOI: 10.1561/1100000005.

[10] J. Dzifcak, M. Scheutz, C. Baral, P. Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of IEEE International Conference on Robotics and Automation*, Kobe, Japan, pp. 4163–4168, 2009. DOI: 10.1109/ROBOT.2009.5152776.

[11] C. Matuszek, E. Herbst, L. Zettlemoyer, D. Fox. Learning to parse natural language commands to a robot control system. In *Proceedings of the 13th International Symposium on Experimental Robotics*, Springer, Heidelberg, Germany, pp. 403–415, 2013. DOI: 10.1007/978-3-319-00065-7_28.

[12] R. F. Ge, R. J. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the 9th Conference on Computational Natural Language Learning*, Association for Computational Linguistics, Ann Arbor, USA, pp. 9–16, 2005. DOI: 10.3115/1706543.1706546.

[13] K. Zhao, L. Huang. Type-driven incremental semantic parsing with polymorphism. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the ACL*, Denver, USA, 2015. DOI: 10.3115/v1/N15-1162.

[14] Y. Artzi, L. Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Edinburgh, UK, pp. 421–432, 2011.

[15] J. Berant, A. Chou, R. Frostig, P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, Washington, USA, 2013.

[16] L. Dong, M. Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, 2016. DOI: 10.18653/v1/P16-1004.

[17] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, 2014. DOI: 10.3115/v1/D14-1181.

[18] Y. N. Dauphin, D. Z. Hakkani-Tur, G. Tur, L. P. Heck. Deep learning for semantic parsing including semantic utterance classification, USA. Patent 20150310862, October 2015.

[19] R. Collobert, J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ACM, Helsinki, Finland, pp. 160–167, 2008. DOI: 10.1145/1390156.1390177.

[20] Z. P. Tu, Z. D. Lu, Y. Liu, X. H. Liu, H. Li. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, 2016. DOI: 10.18653/v1/P16-1008.

[21] A. M. Rush, S. Chopra, J. Weston. A neural attention model for sentence summarization. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015. DOI: 10.18653/v1/D15-1044.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng. ROS: An open-source robot operating system. In *Proceedings of ICRA Workshop on Open Source Software*, Kobe, Japan, 2009.

[23] A. Voutilainen. Part-of-speech tagging. *The Oxford Handbook of Computational Linguistics*, R. Mitkov, Ed., Oxford, UK: Oxford University Press, pp. 219–232, 2003.

[24] D. Q. Chen, C. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Doha, Qatar, 2014.

[25] G. Look, B. Kottahachchi, R. Laddaga, H. Shrobe. A location representation for generating descriptive walking directions. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, ACM, San Diego, USA, pp. 122–129, 2005. DOI: 10.1145/1040830.1040862.

[26] T. Kollar, S. Tellex, D. Roy, N. Roy. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*, Osaka, Japan, pp. 259–266, 2010. DOI: 10.1109/HRI.2010.5453186.

[27] G. Bugmann, E. Klein, S. Lauria, T. Kyriacou. Corpus-based robotics: A route instruction example. In *Proceedings of the 8th International Conference on Intelligent Autonomous Systems*, Amsterdam, Netherlands, pp. 96–103, 2004.

[28] M. MacMahon, B. Stankiewicz, B. Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of National Conference on Artificial Intelligence*, AAAI, Austin, UK, 2006.

[29] P. E. Rybski, J. Stolarz, K. Yoon, M. Veloso. Using dialog and human observations to dictate tasks to a learning robot assistant. *Intelligent Service Robotics*, vol. 1, no. 2, pp. 159–167, 2008. DOI: 10.1007/s11370-008-0016-5.

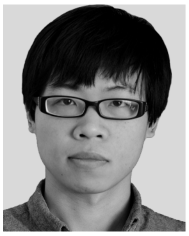
[30] J. Thomason, S. Q. Zhang, R. J. Mooney, P. Stone. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence*, AAAI, Buenos Aires, Argentina, pp. 1923–1929, 2015.

[31] L. B. She, S. H. Yang, Y. Cheng, Y. Y. Jia, J. Y. Chai, N. Xi. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Philadelphia, USA, pp. 89–97, 2014.

[32] E. Grefenstette, P. Blunsom, N. de Freitas, K. M. Hermann. A deep architecture for semantic parsing. In *Proceedings of the ACL Workshop on Semantic Parsing*, Association for Computational Linguistics, Baltimore, USA, 2014.

[33] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami. The limitations of deep learning in adversarial settings. In *Proceedings of IEEE European Symposium on Security and Privacy*, Saarbrucken, Germany, pp. 372–387, 2016. DOI: 10.1109/EuroSP.2016.36.

[34] B. Biggio, B. Nelson, P. Laskov. Support vector machines under adversarial label noise. In *Proceedings of the 3rd Asian Conference on Machine Learning*, Taoyuan, China, pp. 97–112, 2011.



Hao Li received the B. Sc. degree in computer science and technology from Xidian University, China in 2012. He is now a Ph.D. degree candidate at Tsinghua University, China under the supervision of professor Shi-Min Hu. His work has been published in journals including *Communications in Information and Systems, International Journal of Software Engineering and Knowledge Engineering.*

His research interests include program synthesis and system reliability.

E-mail: roselonelh@gmail.com

ORCID iD: 0000-0003-3606-9969

Yu-Ping Wang received the Ph. D. degree in computer science and technology from Tsinghua University, China in 2009. He is currently an associate professor of Tsinghua University, China. He has published papers in important journals and conferences, including *IEEE Transactions on Visualization and Computer Graphics, IEEE Transactions on Computers, Journal of Systems and Software, USENIX Annual Technical Conference, International Symposium on Code Generation and Optimization, International Symposium on Software Reliability Engineering, IEEE International Conference on Computers, Software and Applications (COMPSAC)* and *Asia-Pacific Software Engineering Conference.* He received the COMPSAC 2014 Best Paper Award.

His research interests include robotic system and system reliability.

E-mail: wyp@tsinghua.edu.cn (Corresponding author)

ORCID iD: 0000-0003-4129-7704

Tai-Jiang Mu received the B. Sc. and Ph. D. degrees in computer science and technology from Tsinghua University, China in 2011 and 2016, respectively. He is currently a postdoctoral researcher in Department of Computer Science and Technology, Tsinghua University, China.

His research interests include computer graphics, image/video processing and human-robot interaction.

E-mail: dejungle@mail.tsinghua.edu.cn